scArchest

Marco Wagenstetter, Mohammad Lotfollahi, Mohsen Naghipourfar

Dec 15, 2020

MAIN

scArches is a package to integrate newly produced single-cell datasets into integrated references atlases. Our method can facilitate large collaborative projects with decentralised training and integration of multiple datasets by different groups. scArches is compatible with scanpy. and hosts efficient implementations of all conditional generative models for single-cell data.

CHAPTER

ONE

WHAT CAN YOU DO WITH SCARCHES?

- Construct single or multi-modal (CITE-seq) reference atlases and share the trained model and the data (if possible).
- Download a pre-trained model for your atlas of interest, update it wih new datasets and share with your collaborators.
- Project and integrate query datasets on the top of a reference and use latent representation for downstream tasks, e.g.:diff testing, clustering, classification

CHAPTER

TWO

WHAT ARE DIFFERENT MODELS?

scArches is itself and algorithm to map to project query on the top of reference datasets and is applicable to different models. Here we provide a short explanation and hints when to use which model. Our models are divided into three categories:

CHAPTER

THREE

WHAT ARE DIFFERENT MODELS?

scArches is itself and algorithm to map to project query on the top of reference datasets and is applicable to different models. Here we provide a short explanation and hints when to use which model. Our models are divided into three categories:

Unsupervised This class of algortihms need no *cell type* labels, meaning that you can creat a reference and project a query without having access to cell type labeles. We implemented two algorithms:

- **scVI** (Lopez et al.,2018.): Requires access to raw counts values for data integration and assumes

count distribution on the data (NB, ZINB, Poission).

- **trVAE** (Lotfollahi et al.,2019.): It supports both normalized log tranformed or count data as input and applies additional MMD loss to have better mearging in the latent space.

Supervised and Semi-supervised This class of algorithmes assume the user has access to *cell type* labels when creating the reference data and usaully perfomr better integration compared to. unsupervised methods. However, the query data still can be unlabaled. In addition to integration , you can classify your query cells using these methods.

- **scANVI** (Xu et al.,2019.): It neeeds cell type labels for reference data. Your query data can be either unlabeled or labeled. In case of unlabeled query data you can use this method to also classify your query cells using reference labels.

Multi-modal These algorithms can be used to contstruct multi-modal references atlas and map query data from either modalities on the top of the reference.

- **totalVI** (Gayoso al.,2019.): This model can be used to build multi-modal CITE-seq reference atalses. Query datasets can be either from sc-RNAseq or CITE-seq. In addition to integrating query with reference one can use this model to impute the Proteins in the query datasets.

CHAPTER

FOUR

WHERE TO START?

To get a sense of how the model works please go through this tutorial. To find out how to construct and share or use pre-trained models example sections. Check this example to learn how to start with a raw data and pre-process data for the model.

CHAPTER

FIVE

REFERENCE

If scArches is useful in your research, please consider to cite the preprint.

5.1 scArches (PyTorch) - single-cell architecture surgery

scArches is a package to integrate newly produced single-cell datasets into integrated references atlases. Our method can facilitate large collaborative projects with decentralised training and integration of multiple datasets by different groups. scArches is compatible with scanpy. and hosts efficient implementations of all conditional generative models for single-cell data.

5.1.1 What can you do with scArches?

- Construct single or multi-modal (CITE-seq) reference atlases and share the trained model and the data (if possible).
- Download a pre-trained model for your atlas of interest, update it wih new datasets and share with your collaborators.
- Project and integrate query datasets on the top of a reference and use latent representation for downstream tasks, e.g.:diff testing, clustering, classification

5.1.2 What are different models?

scArches is itself and algorithm to map to project query on the top of reference datasets and is applicable to different models. Here we provide a short explanation and hints when to use which model. Our models are divided into three categories:

5.1.3 What are different models?

scArches is itself and algorithm to map to project query on the top of reference datasets and is applicable to different models. Here we provide a short explanation and hints when to use which model. Our models are divided into three categories:

Unsupervised This class of algortihms need no *cell type* labels, meaning that you can creat a reference and project a query without having access to cell type labeles. We implemented two algorithms:

- **scVI** (Lopez et al.,2018.): Requires access to raw counts values for data integration and assumes

count distribution on the data (NB, ZINB, Poission).

- **trVAE** (Lotfollahi et al.,2019.): It supports both normalized log tranformed or count data as input and applies additional MMD loss to have better mearging in the latent space.

Supervised and Semi-supervised This class of algorithmes assume the user has access to *cell type* labels when creating the reference data and usaully perfomr better integration compared to. unsupervised methods. However, the query data still can be unlabaled. In addition to integration , you can classify your query cells using these methods.

- **scANVI** (Xu et al.,2019.): It neeeds cell type labels for reference data. Your query data can be either unlabeled or labeled. In case of unlabeled query data you can use this method to also classify your query cells using reference labels.

Multi-modal These algorithms can be used to contstruct multi-modal references atlas and map query data from either modalities on the top of the reference.

- **totalVI** (Gayoso al.,2019.): This model can be used to build multi-modal CITE-seq reference atalses. Query datasets can be either from sc-RNAseq or CITE-seq. In addition to integrating query with reference one can use this model to impute the Proteins in the query datasets.

5.1.4 Where to start?

To get a sense of how the model works please go through this tutorial. To find out how to construct and share or use pre-trained models example sections. Check this example to learn how to start with a raw data and pre-process data for the model.

5.1.5 Reference

If scArches is useful in your research, please consider to cite the preprint.

5.2 Installation

scArches requires Python 3.6 or 3.7. We recommend to use Miniconda.

5.2.1 PyPI

The easiest way to get scArches is through pip using the following command:

```
sudo pip install -U scarchest
```

5.2.2 Development

You can also get the latest development version of scArches from Github using the following steps: First, clone scArches using `git`:

```
git clone https://github.com/theislab/scarchesp
```

Then, `cd` to the scArches folder and run the install command:

```
cd scarches
python3 setup.py install
```

On Windows machines you may need to download a C++ compiler if you wish to build from source yourself.

5.2.3 Dependencies

The list of dependencies for scArches can be found in the requirements.txt file in the repository.

If you run into issues, do not hesitate to approach us or raise a GitHub issue.

5.3 API

The API reference contains detailed descriptions of the different end-user classes, functions, methods, etc.

Note: This API reference only contains end-user documentation. If you are looking to hack away at scArches' internals, you will find more detailed comments in the source code.

Import scArchest as:

```
import scarchest as sca
```

After reading the data (`sca.data.read`), you can you can instantiate one of the implemented models from `sca.models` module (currently we support `trVAE`, `scVI`, `scANVI`, and `TotalVI`) and train it on your dataset.

5.3.1 Data Processing

`scarchest.dataset.`**`label_encoder`**(*adata*, *encoder=None*, *condition_key='condition'*)

Encode labels of Annotated *adata* matrix. :param adata: Annotated data matrix. :type adata: : *~anndata.AnnData* :param encoder: dictionary of encoded labels. if *None*, will create one. :type encoder: Dict or None :param condition_key: column name of conditions in *adata.obs* data frame. :type condition_key: String

Returns

- **labels** (*~numpy.ndarray*) – Array of encoded labels
- **label_encoder** (*Dict*) – dictionary with labels and encoded labels as key, value pairs.

`scarchest.dataset.`**`remove_sparsity`**(*adata*)

If `adata.X` is a sparse matrix, this will convert it in to normal matrix. :param adata: Annotated data matrix. :type adata: `AnnData`

Returns **adata** – Annotated dataset.

Return type `AnnData`

`scarchest.dataset.`**`setup_anndata`**(*adata: anndata._core.anndata.AnnData*, *batch_key: Optional[str] = None*, *labels_key: Optional[str] = None*, *layer: Optional[str] = None*, *protein_expression_obsm_key: Optional[str] = None*, *protein_names_uns_key: Optional[str] = None*, *categorical_covariate_keys: Optional[List[str]] = None*, *continuous_covariate_keys: Optional[List[str]] = None*, *copy: bool = False*) → Optional[anndata._core.anndata.AnnData]

Sets up `AnnData` object for *scvi* models.

A mapping will be created between data fields used by *scvi* to their respective locations in adata. This method will also compute the log mean and log variance per batch for the library size prior.

None of the data in adata are modified. Only adds fields to adata.

Parameters

- **adata** – AnnData object containing raw counts. Rows represent cells, columns represent features.
- **batch_key** – key in *adata.obs* for batch information. Categories will automatically be converted into integer categories and saved to *adata.obs['_scvi_batch']*. If *None*, assigns the same batch to all the data.
- **labels_key** – key in *adata.obs* for label information. Categories will automatically be converted into integer categories and saved to *adata.obs['_scvi_labels']*. If *None*, assigns the same label to all the data.
- **layer** – if not *None*, uses this as the key in *adata.layers* for raw count data.
- **protein_expression_obsm_key** – key in *adata.obsm* for protein expression data, Required for `TOTALVI`.
- **protein_names_uns_key** – key in *adata.uns* for protein names. If None, will use the column names of *adata.obsm[protein_expression_obsm_key]* if it is a DataFrame, else will assign sequential names to proteins. Only relevant but not required for `TOTALVI`.
- **categorical_covariate_keys** – keys in *adata.obs* that correspond to categorical data. Used in some *scvi* models.
- **continuous_covariate_keys** – keys in *adata.obs* that correspond to continuous data. Used in some *scvi* models.
- **copy** – if *True*, a copy of adata is returned.

Returns

- If `copy`, will return `AnnData`.
- *Adds the following fields to adata*
- *.uns['_scvi']* – *scvi* setup dictionary
- *.obs['_local_l_mean']* – per batch library size mean
- *.obs['_local_l_var']* – per batch library size variance
- *.obs['_scvi_labels']* – labels encoded as integers
- *.obs['_scvi_batch']* – batch encoded as integers

Examples

Example setting up a scanpy dataset with random gene data and no batch nor label information

```
>>> import scanpy as sc
>>> import scvi
>>> import numpy as np
>>> adata = scvi.data.synthetic_iid(run_setup_anndata=False)
>>> adata
AnnData object with n_obs × n_vars = 400 × 100
    obs: 'batch', 'labels'
    uns: 'protein_names'
    obsm: 'protein_expression'
```

Filter cells and run preprocessing before *setup_anndata*

```
>>> sc.pp.filter_cells(adata, min_counts = 0)
```

Since no batch_key nor labels_key was passed, setup_anndata() will assume all cells have the same batch and label

```
>>> scvi.data.setup_anndata(adata)
INFO      No batch_key inputted, assuming all cells are same batch
INFO      No label_key inputted, assuming all cells have same label
INFO      Using data from adata.X
INFO      Computing library size prior per batch
INFO      Registered keys:['X', 'batch_indices', 'local_l_mean', 'local_l_var',
↪'labels']
INFO      Successfully registered anndata object containing 400 cells, 100 vars,␣
↪1 batches, 1 labels, and 0 proteins. Also registered 0 extra categorical␣
↪covariates and 0 extra continuous covariates.
```

Example setting up scanpy dataset with random gene data, batch, and protein expression

```
>>> adata = scvi.data.synthetic_iid(run_setup_anndata=False)
>>> scvi.data.setup_anndata(adata, batch_key='batch', protein_expression_obsm_key=
↪'protein_expression')
INFO      Using batches from adata.obs["batch"]
INFO      No label_key inputted, assuming all cells have same label
INFO      Using data from adata.X
INFO      Computing library size prior per batch
INFO      Using protein expression from adata.obsm['protein_expression']
INFO      Generating sequential protein names
INFO      Registered keys:['X', 'batch_indices', 'local_l_mean', 'local_l_var',
↪'labels', 'protein_expression']
INFO      Successfully registered anndata object containing 400 cells, 100 vars,␣
↪2 batches, 1 labels, and 100 proteins. Also registered 0 extra categorical␣
↪covariates and 0 extra continuous covariates.
```

scarchest.dataset.**trVAEDataset**
alias of scarchest.dataset.trvae.anndata.AnnotatedDataset

5.3.2 Models

- *trVAE*
- *scVI*
- *scANVI*
- *TotalVI*

trVAE

class scarchest.models.**TRVAE**(*adata: anndata._core.anndata.AnnData*, *condition_key: str = None*, *conditions: Optional[list] = None*, *hidden_layer_sizes: list = [256, 64]*, *latent_dim: int = 10*, *dr_rate: float = 0.05*, *use_mmd: bool = True*, *mmd_on: str = 'z'*, *mmd_boundary: Optional[int] = None*, *recon_loss: Optional[str] = 'nb'*, *beta: float = 1*, *use_bn: bool = False*, *use_ln: bool = True*)

Bases: scarchest.models.trvae.trvae_model.BaseMixin

Model for scArches class. This class contains the implementation of Conditional Variational Auto-encoder.

Parameters

- **adata** (: *~anndata.AnnData*) – Annotated data matrix.
- **condition_key** (*String*) – column name of conditions in *adata.obs* data frame.
- **conditions** (*List*) – List of Condition names that the used data will contain to get the right encoding when used after reloading.
- **hidden_layer_sizes** (*List*) – A list of hidden layer sizes for encoder network. Decoder network will be the reversed order.
- **latent_dim** (*Integer*) – Bottleneck layer (z) size.
- **dr_rate** (*Float*) – Dropput rate applied to all layers, if `dr_rate`==0 no dropout will be applied.
- **use_mmd** (*Boolean*) – If 'True' an additional MMD loss will be calculated on the latent dim. 'z' or the first decoder layer 'y'.
- **mmd_on** (*String*) – Choose on which layer MMD loss will be calculated on if 'use_mmd=True': 'z' for latent dim or 'y' for first decoder layer.
- **mmd_boundary** (*Integer or None*) – Choose on how many conditions the MMD loss should be calculated on. If 'None' MMD will be calculated on all conditions.
- **recon_loss** (*String*) – Definition of Reconstruction-Loss-Method, 'mse', 'nb' or 'zinb'.
- **beta** (*Float*) – Scaling Factor for MMD loss
- **use_bn** (*Boolean*) – If *True* batch normalization will be applied to layers.
- **use_ln** (*Boolean*) – If *True* layer normalization will be applied to layers.

Methods

get_latent([x, c, mean])	Map *x* in to the latent space.
load(dir_path[, adata])	Instantiate a model from the saved output.
load_query_data(adata, reference_model[, ...])	Transfer Learning function for new data.
save(dir_path[, overwrite, save_anndata])	Save the state of the model.
train([n_epochs, lr, eps])	Train the model.

get_latent (*x: Optional[numpy.ndarray] = None*, *c: Optional[numpy.ndarray] = None*, *mean: bool = False*)

Map *x* in to the latent space. This function will feed data in encoder and return z for each sample in data.

Parameters

- **x** – Numpy nd-array to be mapped to latent space. *x* has to be in shape [n_obs, input_dim]. If None, then *self.adata.X* is used.
- **c** – *numpy nd-array* of original (unencoded) desired labels for each sample.
- **mean** – return mean instead of random sample from the latent space

Returns

Return type Returns array containing latent space encoding of 'x'.

classmethod load_query_data (*adata: anndata._core.anndata.AnnData*, *reference_model: Union[str, TRVAE]*, *freeze: bool = True*, *freeze_expression: bool = True*, *remove_dropout: bool = True*)

Transfer Learning function for new data. Uses old trained model and expands it for new conditions.

Parameters

- **adata** – Query anndata object.
- **reference_model** – TRVAE model to expand or a path to TRVAE model folder.
- **freeze** (*Boolean*) – If 'True' freezes every part of the network except the first layers of encoder/decoder.
- **freeze_expression** (*Boolean*) – If 'True' freeze every weight in first layers except the condition weights.
- **remove_dropout** (*Boolean*) – If 'True' remove Dropout for Transfer Learning.

Returns **new_model** – New TRVAE model to train on query data.

Return type trVAE

train(*n_epochs: int = 400*, *lr: float = 0.001*, *eps: float = 0.01*, ***kwargs*)

Train the model.

Parameters

- **n_epochs** – Number of epochs for training the model.
- **lr** – Learning rate for training the model.
- **eps** – torch.optim.Adam eps parameter
- **kwargs** – kwargs for the TrVAE trainer.

scVI

class scarchest.models.**SCVI**(*adata: anndata._core.anndata.AnnData*, *n_hidden: int = 128*, *n_latent: int = 10*, *n_layers: int = 1*, *dropout_rate: float = 0.1*, *dispersion: typing_extensions.Literal['gene', 'gene-batch', 'gene-label', 'gene-cell'] = 'gene'*, *gene_likelihood: typing_extensions.Literal['zinb', 'nb', 'poisson'] = 'zinb'*, *latent_distribution: typing_extensions.Literal['normal', 'ln'] = 'normal'*, *use_cuda: bool = True*, ***model_kwargs*)

Bases: scvi.core.models.rnamixin.RNASeqMixin, scvi.core.models.vaemixin.VAEMixin, scvi.core.models.archesmixin.ArchesMixin, scvi.core.models.base.BaseModelClass

single-cell Variational Inference [Lopez18].

Parameters

- **adata** – AnnData object that has been registered via `setup_anndata()`.
- **n_hidden** – Number of nodes per hidden layer.
- **n_latent** – Dimensionality of the latent space.
- **n_layers** – Number of hidden layers used for encoder and decoder NNs.
- **dropout_rate** – Dropout rate for neural networks.
- **dispersion** – One of the following:
  - `'gene'` - dispersion parameter of NB is constant per gene across cells
  - `'gene-batch'` - dispersion can differ between different batches

  - `'gene-label'` - dispersion can differ between different labels
  - `'gene-cell'` - dispersion can differ for every gene in every cell
- **gene_likelihood** – One of:
  - `'nb'` - Negative binomial distribution
  - `'zinb'` - Zero-inflated negative binomial distribution
  - `'poisson'` - Poisson distribution
- **latent_distribution** – One of:
  - `'normal'` - Normal distribution
  - `'ln'` - Logistic normal distribution (Normal(0, I) transformed by softmax)
- **use_cuda** – Use the GPU or not.
- ****model_kwargs** – Keyword args for `VAE`

Examples

```
>>> adata = anndata.read_h5ad(path_to_anndata)
>>> scvi.data.setup_anndata(adata, batch_key="batch")
>>> vae = scvi.model.SCVI(adata)
>>> vae.train()
>>> adata.obsm["X_scVI"] = vae.get_latent_representation()
>>> adata.obsm["X_normalized_scVI"] = vae.get_normalized_expression()
```

Attributes

history Returns computed metrics during training.

is_trained

test_indices

train_indices

validation_indices

Methods

`differential_expression`([adata, groupby, ...])	A unified method for differential expression analysis.
`get_elbo`([adata, indices, batch_size])	Return the ELBO for the data.
`get_feature_correlation_matrix`([adata, ...])	Generate gene-gene correlation matrix using scvi uncertainty and expression.
`get_latent_library_size`([adata, indices, ...])	Returns the latent library size for each cell.
`get_latent_representation`([adata, indices, ...])	Return the latent representation for each cell.
`get_likelihood_parameters`([adata, indices, ...])	Estimates for the parameters of the likelihood $p(x \mid z)$
`get_marginal_ll`([adata, indices, ...])	Return the marginal LL for the data.

continues on next page

Table 2 – continued from previous page

`get_normalized_expression`([adata, indices, ...])	Returns the normalized (decoded) gene expression.
`get_reconstruction_error`([adata, indices, ...])	Return the reconstruction error for the data.
`load`(dir_path[, adata, use_cuda])	Instantiate a model from the saved output.
`load_query_data`(adata, reference_model[, ...])	Online update of a reference model with scArches algorithm [Lotfollahi20].
`posterior_predictive_sample`([adata, ...])	Generate observation samples from the posterior predictive distribution.
`save`(dir_path[, overwrite, save_anndata])	Save the state of the model.
`train`([n_epochs, train_size, test_size, lr, ...])	Trains the model using amortized variational inference.

scANVI

class scarchest.models.**SCANVI**(*adata: anndata._core.anndata.AnnData*, *unlabeled_category: Union[str, int, float]*, *pretrained_model: Optional[scvi.model.scvi.SCVI] = None*, *n_hidden: int = 128*, *n_latent: int = 10*, *n_layers: int = 1*, *dropout_rate: float = 0.1*, *dispersion: typing_extensions.Literal['gene', 'gene-batch', 'gene-label', 'gene-cell'] = 'gene'*, *gene_likelihood: typing_extensions.Literal['zinb', 'nb', 'poisson'] = 'zinb'*, *use_cuda: bool = True*, ***model_kwargs*)

Bases: `scvi.core.models.rnamixin.RNASeqMixin`, `scvi.core.models.vaemixin.VAEMixin`, `scvi.core.models.archesmixin.ArchesMixin`, `scvi.core.models.base.BaseModelClass`

Single-cell annotation using variational inference [Xu19].

Inspired from M1 + M2 model, as described in (https://arxiv.org/pdf/1406.5298.pdf).

Parameters

- **adata** – AnnData object that has been registered via `setup_anndata()`.
- **unlabeled_category** – Value used for unlabeled cells in *labels_key* used to setup AnnData with scvi.
- **pretrained_model** – Instance of SCVI model that has already been trained.
- **n_hidden** – Number of nodes per hidden layer.
- **n_latent** – Dimensionality of the latent space.
- **n_layers** – Number of hidden layers used for encoder and decoder NNs.
- **dropout_rate** – Dropout rate for neural networks.
- **dispersion** – One of the following:
  - `'gene'` - dispersion parameter of NB is constant per gene across cells
  - `'gene-batch'` - dispersion can differ between different batches
  - `'gene-label'` - dispersion can differ between different labels
  - `'gene-cell'` - dispersion can differ for every gene in every cell
- **gene_likelihood** – One of:

- `'nb'` - Negative binomial distribution
- `'zinb'` - Zero-inflated negative binomial distribution
- `'poisson'` - Poisson distribution

- **use_cuda** – Use the GPU or not.
- ****model_kwargs** – Keyword args for `VAE` and `SCANVAE`

Examples

```
>>> adata = anndata.read_h5ad(path_to_anndata)
>>> scvi.data.setup_anndata(adata, batch_key="batch", labels_key="labels")
>>> vae = scvi.model.SCANVI(adata, "Unknown")
>>> vae.train()
>>> adata.obsm["X_scVI"] = vae.get_latent_representation()
>>> adata.obs["pred_label"] = vae.predict()
```

Attributes

history Returns computed metrics during training.

is_trained

test_indices

train_indices

validation_indices

Methods

`differential_expression`([adata, groupby, ...])	A unified method for differential expression analysis.
`get_elbo`([adata, indices, batch_size])	Return the ELBO for the data.
`get_feature_correlation_matrix`([adata, ...])	Generate gene-gene correlation matrix using scvi uncertainty and expression.
`get_latent_library_size`([adata, indices, ...])	Returns the latent library size for each cell.
`get_latent_representation`([adata, indices, ...])	Return the latent representation for each cell.
`get_likelihood_parameters`([adata, indices, ...])	Estimates for the parameters of the likelihood $p(x \mid z)$
`get_marginal_ll`([adata, indices, ...])	Return the marginal LL for the data.
`get_normalized_expression`([adata, indices, ...])	Returns the normalized (decoded) gene expression.
`get_reconstruction_error`([adata, indices, ...])	Return the reconstruction error for the data.
`load`(dir_path[, adata, use_cuda])	Instantiate a model from the saved output.
`load_query_data`(adata, reference_model[, ...])	Online update of a reference model with scArches algorithm [Lotfollahi20].
`posterior_predictive_sample`([adata, ...])	Generate observation samples from the posterior predictive distribution.
`predict`([adata, indices, soft, batch_size])	Return cell label predictions.

continues on next page

Table 3 – continued from previous page

save(dir_path[, overwrite, save_anndata])	Save the state of the model.
train([n_epochs_unsupervised, ...])	Train the model.

property history
Returns computed metrics during training.

predict(*adata: Optional[anndata._core.anndata.AnnData] = None*, *indices: Optional[Sequence[int]] = None*, *soft: bool = False*, *batch_size: int = 128*) → Union[numpy.ndarray, pandas.core.frame.DataFrame]
Return cell label predictions.

Parameters

- **adata** – AnnData object that has been registered via `setup_anndata()`.
- **indices** – Indices of cells in adata to use. If *None*, all cells are used.
- **soft** – Return probabilities for each class label.
- **batch_size** – Minibatch size to use.

train(*n_epochs_unsupervised: Optional[int] = None*, *n_epochs_semisupervised: Optional[int] = None*, *train_base_model: bool = True*, *train_size: float = 0.9*, *test_size: float = None*, *lr: float = 0.001*, *n_epochs_kl_warmup: int = 400*, *n_iter_kl_warmup: Optional[int] = None*, *frequency: Optional[int] = None*, *unsupervised_trainer_kwargs: dict = {}*, *semisupervised_trainer_kwargs: dict = {}*, *unsupervised_train_kwargs: dict = {}*, *semisupervised_train_kwargs: dict = {}*)
Train the model.

Parameters

- **n_epochs_unsupervised** – Number of passes through the dataset for unsupervised pre-training.
- **n_epochs_semisupervised** – Number of passes through the dataset for semisupervised training.
- **train_base_model** – Pretrain an SCVI base model first before semisupervised training.
- **train_size** – Size of training set in the range [0.0, 1.0].
- **test_size** – Size of the test set. If *None*, defaults to 1 - *train_size*. If *train_size* + *test_size* < *1*, the remaining cells belong to a validation set.
- **lr** – Learning rate for optimization.
- **n_epochs_kl_warmup** – Number of passes through dataset for scaling term on KL divergence to go from 0 to 1.
- **n_iter_kl_warmup** – Number of minibatches for scaling term on KL divergence to go from 0 to 1. To use, set to not *None* and set *n_epochs_kl_warmup* to *None*.
- **frequency** – Frequency with which metrics are computed on the data for train/test/val sets for both the unsupervised and semisupervised trainers. If you'd like a different frequency for the semisupervised trainer, set frequency in semisupervised_train_kwargs.
- **unsupervised_trainer_kwargs** – Other keyword args for `UnsupervisedTrainer`.
- **semisupervised_trainer_kwargs** – Other keyword args for `SemiSupervisedTrainer`.

- **semisupervised_train_kwargs** – Keyword args for the train method of `SemiSupervisedTrainer`.

TotalVI

class scarchest.models.**TOTALVI**(*adata: anndata._core.anndata.AnnData*, *n_latent: int = 20*, *gene_dispersion: typing_extensions.Literal['gene', 'gene-batch', 'gene-label', 'gene-cell'] = 'gene'*, *protein_dispersion: typing_extensions.Literal['protein', 'protein-batch', 'protein-label'] = 'protein'*, *gene_likelihood: typing_extensions.Literal['zinb', 'nb'] = 'nb'*, *latent_distribution: typing_extensions.Literal['normal', 'ln'] = 'normal'*, *empirical_protein_background_prior: bool = True*, *use_cuda: bool = True*, ***model_kwargs*)

Bases: `scvi.core.models.rnamixin.RNASeqMixin`, `scvi.core.models.vaemixin.VAEMixin`, `scvi.core.models.archesmixin.ArchesMixin`, `scvi.core.models.base.BaseModelClass`

total Variational Inference [GayosoSteier20].

Parameters

- **adata** – AnnData object that has been registered via `setup_anndata()`.
- **n_latent** – Dimensionality of the latent space.
- **gene_dispersion** – One of the following:
  - `'gene'` - genes_dispersion parameter of NB is constant per gene across cells
  - `'gene-batch'` - genes_dispersion can differ between different batches
  - `'gene-label'` - genes_dispersion can differ between different labels
- **protein_dispersion** – One of the following:
  - `'protein'` - protein_dispersion parameter is constant per protein across cells
  - `'protein-batch'` - protein_dispersion can differ between different batches NOT TESTED
  - `'protein-label'` - protein_dispersion can differ between different labels NOT TESTED
- **gene_likelihood** – One of:
  - `'nb'` - Negative binomial distribution
  - `'zinb'` - Zero-inflated negative binomial distribution
- **latent_distribution** – One of:
  - `'normal'` - Normal distribution
  - `'ln'` - Logistic normal distribution (Normal(0, I) transformed by softmax)
- **empirical_protein_background_prior** – Set the initialization of protein background prior empirically. This option fits a GMM for each of 100 cells per batch and averages the distributions. Note that even with this option set to *True*, this only initializes a parameter that is learned during inference. If *False*, randomly initializes.
- **use_cuda** – Use the GPU or not.
- ****model_kwargs** – Keyword args for `TOTALVAE`

Examples

```
>>> adata = anndata.read_h5ad(path_to_anndata)
>>> scvi.data.setup_anndata(adata, batch_key="batch", protein_expression_obsm_key=
↪"protein_expression")
>>> vae = scvi.model.TOTALVI(adata)
>>> vae.train()
>>> adata.obsm["X_totalVI"] = vae.get_latent_representation()
```

Attributes

history Returns computed metrics during training.

is_trained

test_indices

train_indices

validation_indices

Methods

differential_expression([adata, groupby, ...])	A unified method for differential expression analysis.
get_elbo([adata, indices, batch_size])	Return the ELBO for the data.
get_feature_correlation_matrix([adata, ...])	Generate gene-gene correlation matrix using scvi uncertainty and expression.
get_latent_library_size([adata, indices, ...])	Returns the latent library size for each cell.
get_latent_representation([adata, indices, ...])	Return the latent representation for each cell.
get_likelihood_parameters([adata, indices, ...])	Estimates for the parameters of the likelihood $p(x, y \mid z)$.
get_marginal_ll([adata, indices, ...])	Return the marginal LL for the data.
get_normalized_expression([adata, indices, ...])	Returns the normalized gene expression and protein expression.
get_protein_foreground_probability([adata, ...])	Returns the foreground probability for proteins.
get_reconstruction_error([adata, indices, ...])	Return the reconstruction error for the data.
load(dir_path[, adata, use_cuda])	Instantiate a model from the saved output.
load_query_data(adata, reference_model[, ...])	Online update of a reference model with scArches algorithm [Lotfollahi20].
posterior_predictive_sample([adata, ...])	Generate observation samples from the posterior predictive distribution.
save(dir_path[, overwrite, save_anndata])	Save the state of the model.
train([n_epochs, train_size, test_size, lr, ...])	Train the model.

differential_expression(*adata: Optional[anndata._core.anndata.AnnData] = None*, *groupby: Optional[str] = None*, *group1: Optional[Iterable[str]] = None*, *group2: Optional[str] = None*, *idx1: Optional[Union[Sequence[int], Sequence[bool]]] = None*, *idx2: Optional[Union[Sequence[int], Sequence[bool]]] = None*, *mode: typing_extensions.Literal['vanilla', 'change'] = 'change'*, *delta: float = 0.25*, *batch_size: Optional[int] = None*, *all_stats: bool = True*, *batch_correction: bool = False*, *batchid1: Optional[Iterable[str]] = None*, *batchid2: Optional[Iterable[str]] = None*, *fdr_target: float = 0.05*, *protein_prior_count: float = 0.1*, *scale_protein: bool = False*, *sample_protein_mixing: bool = False*, *include_protein_background: bool = False*, ***kwargs*) → pandas.core.frame.DataFrame

A unified method for differential expression analysis.

Implements *"vanilla"* DE [Lopez18] and *"change"* mode DE [Boyeau19].

Parameters

- **adata** – AnnData object with equivalent structure to initial AnnData. If None, defaults to the AnnData object used to initialize the model.
- **groupby** – The key of the observations grouping to consider.
- **group1** – Subset of groups, e.g. [*'g1'*, *'g2'*, *'g3'*], to which comparison shall be restricted, or all groups in *groupby* (default).
- **group2** – If *None*, compare each group in *group1* to the union of the rest of the groups in *groupby*. If a group identifier, compare with respect to this group.
- **idx1** – Boolean mask or indices for *group1*. *idx1* and *idx2* can be used as an alternative to the AnnData keys. If *idx1* is not *None*, this option overrides *group1* and *group2*.
- **idx2** – Boolean mask or indices for *group2*. By default, includes all cells not specified in *idx1*.
- **mode** – Method for differential expression. See user guide for full explanation.
- **delta** – specific case of region inducing differential expression. In this case, we suppose that $R \setminus [-\delta, \delta]$ does not induce differential expression (change model default case).
- **batch_size** – Minibatch size for data loading into model. Defaults to *scvi.settings.batch_size*.
- **all_stats** – Concatenate count statistics (e.g., mean expression group 1) to DE results.
- **batch_correction** – Whether to correct for batch effects in DE inference.
- **batchid1** – Subset of categories from *batch_key* registered in `setup_anndata()`, e.g. [*'batch1'*, *'batch2'*, *'batch3'*], for *group1*. Only used if *batch_correction* is *True*, and by default all categories are used.
- **batchid2** – Same as *batchid1* for group2. *batchid2* must either have null intersection with *batchid1*, or be exactly equal to *batchid1*. When the two sets are exactly equal, cells are compared by decoding on the same batch. When sets have null intersection, cells from *group1* and *group2* are decoded on each group in *group1* and *group2*, respectively.
- **fdr_target** – Tag features as DE based on posterior expected false discovery rate.
- **protein_prior_count** – Prior count added to protein expression before LFC computation
- **scale_protein** – Force protein values to sum to one in every single cell (post-hoc normalization)

- **sample_protein_mixing** – Sample the protein mixture component, i.e., use the parameter to sample a Bernoulli that determines if expression is from foreground/background.
- **include_protein_background** – Include the protein background component as part of the protein expression
- ****kwargs** – Keyword args for `scvi.core.utils.DifferentialComputation.get_bayes_factors()`

Returns

Return type Differential expression DataFrame.

get_feature_correlation_matrix(*adata=None*, *indices=None*, *n_samples: int = 10*, *batch_size: int = 64*, *rna_size_factor: int = 1000*, *transform_batch: Optional[Sequence[Union[Number, str]]] = None*, *correlation_type: typing_extensions.Literal['spearman', 'pearson'] = 'spearman'*, *log_transform: bool = False*) → pandas.core.frame.DataFrame

Generate gene-gene correlation matrix using scvi uncertainty and expression.

Parameters

- **adata** – AnnData object with equivalent structure to initial AnnData. If *None*, defaults to the AnnData object used to initialize the model.
- **indices** – Indices of cells in adata to use. If *None*, all cells are used.
- **n_samples** – Number of posterior samples to use for estimation.
- **batch_size** – Minibatch size for data loading into model. Defaults to *scvi.settings.batch_size*.
- **rna_size_factor** – size factor for RNA prior to sampling gamma distribution
- **transform_batch** – Batches to condition on. If transform_batch is:
  - None, then real observed batch is used
  - int, then batch transform_batch is used
  - list of int, then values are averaged over provided batches.
- **correlation_type** – One of "pearson", "spearman".
- **log_transform** – Whether to log transform denoised values prior to correlation calculation.

Returns

Return type Gene-protein-gene-protein correlation matrix

get_latent_library_size(*adata: Optional[anndata._core.anndata.AnnData] = None*, *indices: Optional[Sequence[int]] = None*, *give_mean: bool = True*, *batch_size: Optional[int] = None*) → numpy.ndarray

Returns the latent library size for each cell.

This is denoted as ℓ_n in the totalVI paper.

Parameters

- **adata** – AnnData object with equivalent structure to initial AnnData. If *None*, defaults to the AnnData object used to initialize the model.

- **indices** – Indices of cells in adata to use. If *None*, all cells are used.
- **give_mean** – Return the mean or a sample from the posterior distribution.
- **batch_size** – Minibatch size for data loading into model. Defaults to *scvi.settings.batch_size*.

get_latent_representation(*adata: Optional[anndata._core.anndata.AnnData] = None*, *indices: Optional[Sequence[int]] = None*, *give_mean: bool = True*, *mc_samples: int = 5000*, *batch_size: Optional[int] = None*) → numpy.ndarray

Return the latent representation for each cell.

Parameters

- **adata** – AnnData object with equivalent structure to initial AnnData. If *None*, defaults to the AnnData object used to initialize the model.
- **indices** – Indices of cells in adata to use. If *None*, all cells are used.
- **give_mean** – Give mean of distribution or sample from it
- **mc_samples** – For distributions with no closed-form mean (e.g., *logistic normal*), how many Monte Carlo samples to take for computing mean.
- **batch_size** – Minibatch size for data loading into model. Defaults to *scvi.settings.batch_size*.

Returns **latent_representation** – Low-dimensional representation for each cell

Return type np.ndarray

Examples

```
>>> vae = scvi.model.TOTALVI(adata)
>>> vae.train(n_epochs=400)
>>> adata.obsm["X_totalVI"] = vae.get_latent_representation()
```

We can also get the latent representation for a subset of cells

```
>>> adata_subset = adata[adata.obs.cell_type == "really cool cell type"]
>>> latent_subset = vae.get_latent_representation(adata_subset)
```

get_likelihood_parameters(*adata: Optional[anndata._core.anndata.AnnData] = None*, *indices: Optional[Sequence[int]] = None*, *n_samples: Optional[int] = 1*, *give_mean: Optional[bool] = False*, *batch_size: Optional[int] = None*) → Dict[str, numpy.ndarray]

Estimates for the parameters of the likelihood $p(x, y \mid z)$.

Parameters

- **adata** – AnnData object with equivalent structure to initial AnnData. If *None*, defaults to the AnnData object used to initialize the model.
- **indices** – Indices of cells in adata to use. If *None*, all cells are used.
- **n_samples** – Number of posterior samples to use for estimation.
- **give_mean** – Return expected value of parameters or a samples
- **batch_size** – Minibatch size for data loading into model. Defaults to *scvi.settings.batch_size*.

get_normalized_expression(*adata=None*, *indices=None*, *transform_batch: Optional[Sequence[Union[Number, str]]] = None*, *gene_list: Optional[Sequence[str]] = None*, *protein_list: Optional[Sequence[str]] = None*, *library_size: Optional[Union[float, typing_extensions.Literal['latent']]] = 1*, *n_samples: int = 1*, *sample_protein_mixing: bool = False*, *scale_protein: bool = False*, *include_protein_background: bool = False*, *batch_size: Optional[int] = None*, *return_mean: bool = True*, *return_numpy: Optional[bool] = None*) → Tuple[Union[numpy.ndarray, pandas.core.frame.DataFrame], Union[numpy.ndarray, pandas.core.frame.DataFrame]]

Returns the normalized gene expression and protein expression.

This is denoted as ρ_n in the totalVI paper for genes, and TODO for proteins, $(1 - \pi_{nt})\alpha_{nt}\beta_{nt}$.

Parameters

- **adata** – AnnData object with equivalent structure to initial AnnData. If *None*, defaults to the AnnData object used to initialize the model.
- **indices** – Indices of cells in adata to use. If *None*, all cells are used.
- **transform_batch** – Batch to condition on. If transform_batch is:
  - None, then real observed batch is used
  - int, then batch transform_batch is used
  - List[int], then average over batches in list
- **gene_list** – Return frequencies of expression for a subset of genes. This can save memory when working with large datasets and few genes are of interest.
- **protein_list** – Return protein expression for a subset of genes. This can save memory when working with large datasets and few genes are of interest.
- **library_size** – Scale the expression frequencies to a common library size. This allows gene expression levels to be interpreted on a common scale of relevant magnitude.
- **n_samples** – Get sample scale from multiple samples.
- **sample_protein_mixing** – Sample mixing bernoulli, setting background to zero
- **scale_protein** – Make protein expression sum to 1
- **include_protein_background** – Include background component for protein expression
- **batch_size** – Minibatch size for data loading into model. Defaults to *scvi.settings.batch_size*.
- **return_mean** – Whether to return the mean of the samples.
- **return_numpy** – Return a *np.ndarray* instead of a *pd.DataFrame*. Includes gene names as columns. If either n_samples=1 or return_mean=True, defaults to False. Otherwise, it defaults to True.

Returns

- - ***gene_normalized_expression** - normalized expression for RNA*
- - ***protein_normalized_expression** - normalized expression for proteins*
- If `n_samples` > 1 and `return_mean` is False, then the shape is `(samples, cells, genes)`.

- Otherwise, shape is `(cells, genes)`. Return type is `pd.DataFrame` unless `return_numpy` is True.

get_protein_foreground_probability(*adata: Optional[anndata._core.anndata.AnnData] = None*, *indices: Optional[Sequence[int]] = None*, *transform_batch: Optional[Sequence[Union[Number, str]]] = None*, *protein_list: Optional[Sequence[str]] = None*, *n_samples: int = 1*, *batch_size: Optional[int] = None*, *return_mean: bool = True*, *return_numpy: Optional[bool] = None*)

Returns the foreground probability for proteins.

This is denoted as $(1 - \pi_{nt})$ in the totalVI paper.

Parameters

- **adata** – AnnData object with equivalent structure to initial AnnData. If *None*, defaults to the AnnData object used to initialize the model.
- **indices** – Indices of cells in adata to use. If *None*, all cells are used.
- **transform_batch** – Batch to condition on. If transform_batch is:
  - None, then real observed batch is used
  - int, then batch transform_batch is used
  - List[int], then average over batches in list
- **protein_list** – Return protein expression for a subset of genes. This can save memory when working with large datasets and few genes are of interest.
- **n_samples** – Number of posterior samples to use for estimation.
- **batch_size** – Minibatch size for data loading into model. Defaults to *scvi.settings.batch_size*.
- **return_mean** – Whether to return the mean of the samples.
- **return_numpy** – Return a `ndarray` instead of a `DataFrame`. DataFrame includes gene names as columns. If either *n_samples=1* or *return_mean=True*, defaults to *False*. Otherwise, it defaults to *True*.

Returns

- - ***foreground_probability** - probability foreground for each protein*
- If *n_samples* > 1 and *return_mean* is False, then the shape is *(samples, cells, genes)*.
- Otherwise, shape is *(cells, genes)*. In this case, return type is `DataFrame` unless *return_numpy* is True.

get_reconstruction_error(*adata: Optional[anndata._core.anndata.AnnData] = None*, *indices: Optional[Sequence[int]] = None*, *mode: typing_extensions.Literal['total', 'gene', 'protein'] = 'total'*, *batch_size: Optional[int] = None*)

Return the reconstruction error for the data.

This is typically written as $p(x, y \mid z)$, the likelihood term given one posterior sample. Note, this is not the negative likelihood, higher is better.

Parameters

- **adata** – AnnData object with equivalent structure to initial AnnData. If *None*, defaults to the AnnData object used to initialize the model.
- **indices** – Indices of cells in adata to use. If *None*, all cells are used.
- **mode** – Compute for genes, proteins, or both.
- **batch_size** – Minibatch size for data loading into model. Defaults to *scvi.settings.batch_size*.

posterior_predictive_sample(*adata: Optional[anndata._core.anndata.AnnData] = None*, *indices: Optional[Sequence[int]] = None*, *n_samples: int = 1*, *batch_size: Optional[int] = None*, *gene_list: Optional[Sequence[str]] = None*, *protein_list: Optional[Sequence[str]] = None*) → numpy.ndarray

Generate observation samples from the posterior predictive distribution.

The posterior predictive distribution is written as $p(\hat{x}, \hat{y} \mid x, y)$.

Parameters

- **adata** – AnnData object with equivalent structure to initial AnnData. If *None*, defaults to the AnnData object used to initialize the model.
- **indices** – Indices of cells in adata to use. If *None*, all cells are used.
- **n_samples** – Number of required samples for each cell
- **batch_size** – Minibatch size for data loading into model. Defaults to *scvi.settings.batch_size*.
- **gene_list** – Names of genes of interest
- **protein_list** – Names of proteins of interest

Returns **x_new** – tensor with shape (n_cells, n_genes, n_samples)

Return type ndarray

train(*n_epochs: int = 400*, *train_size: float = 0.9*, *test_size: Optional[float] = None*, *lr: float = 0.004*, *n_epochs_kl_warmup: Optional[int] = None*, *n_iter_kl_warmup: Union[typing_extensions.Literal['auto'], int] = 'auto'*, *batch_size: int = 256*, *frequency: Optional[int] = None*, *train_fun_kwargs: dict = {}*, ***kwargs*)

Train the model.

Parameters

- **n_epochs** – Number of passes through the dataset.
- **train_size** – Size of training set in the range [0.0, 1.0].
- **test_size** – Size of the test set. If *None*, defaults to 1 - *train_size*. If *train_size* + *test_size* < *1*, the remaining cells belong to a validation set.
- **lr** – Learning rate for optimization.
- **n_epochs_kl_warmup** – Number of passes through dataset for scaling term on KL divergence to go from 0 to 1.
- **n_iter_kl_warmup** – Number of minibatches for scaling term on KL divergence to go from 0 to 1. To use, set to not *None* and set *n_epochs_kl_warmup* to *None*.
- **batch_size** – Minibatch size to use during training.
- **frequency** – Frequency with which metrics are computed on the data for train/test/val sets.

- **train_fun_kwargs** – Keyword args for the train method of `TotalTrainer`.
- ****kwargs** – Other keyword args for `TotalTrainer`.

5.3.3 Plotting

class scarchest.plotting.**SCVI_EVAL**(*model: Union[scvi.model.scvi.SCVI, scvi.model.scanvi.SCANVI, scvi.model.totalvi.TOTALVI]*, *adata: anndata._core.anndata.AnnData*, *trainer: Union[scarchest.trainers.scvi.trainers.scVITrainer, scarchest.trainers.scvi.trainers.scANVITrainer, scarchest.trainers.scvi.trainers.totalTrainer] = None*, *cell_type_key: str = None*, *batch_key: str = None*)

Bases: `object`

Methods

plot_latent([show, save, dir_path, ...])	if save:

get_asw	
get_classification_accuracy	
get_ebm	
get_f1_score	
get_knn_purity	
get_latent_score	
get_model_arch	
get_nmi	
latent_as_anndata	
plot_history	

get_asw()

get_classification_accuracy()

get_ebm(*n_neighbors=50*, *n_pools=50*, *n_samples_per_pool=100*, *verbose=True*)

get_f1_score()

get_knn_purity(*n_neighbors=50*, *verbose=True*)

get_latent_score()

get_model_arch()

get_nmi()

latent_as_anndata()

plot_history(*show=True*, *save=False*, *dir_path=None*)

plot_latent(*show=True*, *save=False*, *dir_path=None*, *n_neighbors=8*, *predictions=False*, *in_one=False*, *colors=None*)

if save:

if dir_path is None: name = 'scanvi_latent.png'

else: name = f'{dir_path}.png'

else: name = False

class scarchest.plotting.**TRVAE_EVAL**(*model: Union[scarchest.models.trvae.trvae.trVAE, scarchest.models.trvae.trvae_model.TRVAE]*, *adata: anndata._core.anndata.AnnData*, *trainer: scarchest.trainers.trvae.unsupervised.trVAETrainer = None*, *condition_key: str = None*, *cell_type_key: str = None*)

Bases: object

Methods

get_asw	
get_ebm	
get_knn_purity	
get_latent_score	
get_model_arch	
get_nmi	
latent_as_anndata	
plot_history	
plot_latent	

get_asw()

get_ebm(*n_neighbors=50*, *n_pools=50*, *n_samples_per_pool=100*, *verbose=True*)

get_knn_purity(*n_neighbors=50*, *verbose=True*)

get_latent_score()

get_model_arch()

get_nmi()

latent_as_anndata()

plot_history(*show=True*, *save=False*, *dir_path=None*)

plot_latent(*show=True*, *save=False*, *dir_path=None*, *n_neighbors=8*)

scarchest.plotting.**sankey_diagram**(*data*, *save_path=None*, *show=False*, ***kwargs*)

Draws Sankey diagram for the given data. :param data: array with 2 columns. One for predictions and another for true values. :type data: ndarray :param save_path: Path to save the drawn Sankey diagram. if None, the diagram will not be saved. :type save_path: str :param show: if True will show the diagram. :type show: bool :param kwargs: additional arguments for diagram configuration. See _alluvial.plot function.

5.3.4 Zenodo

- *Deposition helpers*
- *File helpers*

scarchest.zenodo.**download_model**(*download_link: str*, *save_path: str = './'*, *make_dir: bool = False*)

Downloads the zip file of the model in the link and saves it in save_path and extracts.

Parameters

- **link** (*str*) – Direct downloadable link.
- **save_path** (*str*) – Directory path for downloaded file
- **make_dir** (*bool*) – Whether to make the `save_path` if it does not exist in the system.

Returns **extract_dir** – Full path to the folder of the model.

Return type str

`scarchest.zenodo.`**upload_model**(*model: Union[scarchest.models.trvae.trvae_model.TRVAE, scvi.model.scvi.SCVI, scvi.model.scanvi.SCANVI, scvi.model.totalvi.TOTALVI, str]*, *deposition_id: str*, *access_token: str*, *model_name: str = None*)

Uploads trained `model` to Zenodo.

Parameters

- **model** (*TRVAE*, *SCVI*, *SCANVI*, *TOTALVI*, str) – An instance of one of classes defined in `scarchest.models` module or a path to a saved model.
- **deposition_id** (*str*) – ID of a deposition in your Zenodo account.
- **access_token** (*str*) – Your Zenodo access token.
- **model_name** (*str*) – An optional name of the model to upload

Returns **download_link** – Generated direct download link for the uploaded model in the deposition. Please **Note** that the link is usable **after** your published your deposition.

Return type str

Deposition helpers

`scarchest.zenodo.deposition.`**create_deposition**(*access_token: str*, *upload_type: str*, *title: str*, *description: str*, ***kwargs*)

Creates a deposition in your Zenodo account.

Parameters

- **access_token** (*str*) – Your Zenodo access token.
- **upload_type** (*str*) –
- **title** (*str*) –
- **description** (*str*) –
- **kwargs** –

Returns **deposition_id** – ID of the created deposition.

Return type str

`scarchest.zenodo.deposition.`**delete_deposition**(*deposition_id: str*, *access_token: str*)

Deletes the existing deposition with `deposition_id` in your Zenodo account.

Parameters

- **deposition_id** (*str*) – ID of a deposition in your Zenodo account.
- **access_token** (*str*) – Your Zenodo Access token.

`scarchest.zenodo.deposition.`**get_all_deposition_ids**(*access_token: str*)

Gets list of all of deposition IDs existed in your Zenodo account.

Parameters **access_token** (*str*) – Your Zenodo access token.

Returns **deposition_ids** – List of deposition IDs.

Return type list

scarchest.zenodo.deposition.**publish_deposition**(*deposition_id: str*, *access_token: str*)
Publishes the existing deposition with `deposition_id` in your Zenodo account.

Parameters

- **deposition_id** (*str*) – ID of a deposition in your Zenodo account.
- **access_token** (*str*) – Your Zenodo access token.

Returns **download_link** – Generated direct download link for the uploaded model in the deposition. Please **Note** that the link is usable **after** your published your deposition.

Return type str

scarchest.zenodo.deposition.**update_deposition**(*deposition_id: str*, *access_token: str*, *metadata: dict*)
Updates the existing deposition with `deposition_id` in your Zenodo account.

Parameters

- **deposition_id** (*str*) – ID of a deposition in your Zenodo account.
- **access_token** (*str*) – Your Zenodo access token.
- **metadata** (*dict*) –

File Helpers

scarchest.zenodo.file.**download_file**(*link: str*, *save_path: str = None*, *make_dir: bool = False*)
Downloads the file in the `link` and saves it in `save_path`.

Parameters

- **link** (*str*) – Direct downloadable link.
- **save_path** (*str*) – Path with the name and extension of downloaded file.
- **make_dir** (*bool*) – Whether to make the `save_path` if it does not exist in the system.

Returns

- **file_path** (*str*) – Full path with name and extension of downloaded file.
- **http_response** (`HTTPMessage`) – `HttpMessage` object containing status code and information about the http request.

scarchest.zenodo.file.**upload_file**(*file_path: str*, *deposition_id: str*, *access_token: str*)
Downloads the file in the `link` and saves it in `save_path`.

Parameters

- **file_path** (*str*) – Full path with the name and extension of the file you want to upload.
- **deposition_id** (*str*) – ID of a deposition in your Zenodo account.
- **access_token** (*str*) – Your Zenodo Access token.

Returns

- **file_path** (*str*) – Full path with name and extension of downloaded file.

- **http_response** (`HTTPMessage`) – `HttpMessage` object containing status code and information about the http request.

5.4 How to share your models?

Once you have uploaded your model according to this toturial, please share the information about the model and the data you have used by filling this form.

5.5 Model database

The list of pretrained models are available in here.

5.6 A few tips on training models

- We recommend you to set *loss_fn = nb* or *zinb*. These loss functions require the access to count and not normalized data. You need to have normalized log-transformed data in *adata.X* and raw count data in *adata.raw.X*. You also need to have normalization factors for each cell in *adata.obs[scale_factors]*. These normalization factors can be obtained with scanpy.pp.normalize_total or other normalization methods such as scran.
- If you don't have access to count data and have normalized data then set *loss_fn* to *sse* or *mse*.
- If you want better separation of cell types you can increase the *n_epochs*. 100 epochs in most cases yield good quality but you can increase uo to 200. If some cell types are merged which should not be try to increase *n_epochs* and decrease *alpha* (see next tip).
- If you want to increase the mixing of the different batches then try to increase *alpha* when tou construct the the model. Maximum value of *alpha* can be 1. Increasing alpha will give you better mixing but it is a trade off! Increase *alpha* might also merge some small cell types or conditions. You can start with very small values (e.g 0.0001) and then increase that (0.001 ->0.005->0.01 and even 0.1 and finally 0.5).
- It is important to use highly variable genes for training. We recommend to use at least 2000 hvgs and if you have more complicated datasets, conditions then try to increase it to 5000 or so to include enough information for the model.
- Regarding *architecture* always try with the default one ([128,128], *z_dimension`=10) and check the results. If you have more complicated data sets with many datasets and conditions and etc then you can increase the depth ([128,128,128] or [128,128,128,128]). According to our experiments small values of `z_dimension* between 10 (default) and 20 are good.

5.7 Unsupervised surgery pipeline with SCVI

[1]:
```
import os
os.chdir('../')
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
warnings.simplefilter(action='ignore', category=UserWarning)
```

[2]:
```
import scanpy as sc
import torch
import scarchest as sca
from scarchest.dataset.trvae.data_handling import remove_sparsity
import matplotlib.pyplot as plt
import numpy as np
import gdown
```

[3]:
```
sc.settings.set_figure_params(dpi=200, frameon=False)
sc.set_figure_params(dpi=200)
sc.set_figure_params(figsize=(4, 4))
torch.set_printoptions(precision=3, sci_mode=False, edgeitems=7)
```

5.7.1 Set relevant anndata.obs labels and training length

Here we use the CelSeq2 and SS2 studies as query data and the other 3 studies as reference atlas. We strongly suggest to use earlystopping to avoid over-fitting. The best earlystopping criteria is the 'elbo' for SCVI.

[4]:
```
condition_key = 'study'
cell_type_key = 'cell_type'
target_conditions = ['Pancreas CelSeq2', 'Pancreas SS2']


vae_epochs = 500
surgery_epochs = 500

early_stopping_kwargs = {
    "early_stopping_metric": "elbo",
    "save_best_state_metric": "elbo",
    "patience": 10,
    "threshold": 0,
    "reduce_lr_on_plateau": True,
    "lr_patience": 8,
    "lr_factor": 0.1,
}
```

5.7.2 Download Dataset and split into reference dataset and query dataset

[5]:
```
url = 'https://drive.google.com/uc?id=1ehxgfHTsMZXy6YzlFKGJOsBKQ5rrvMnd'
output = 'pancreas.h5ad'
gdown.download(url, output, quiet=False)
```

```
Downloading...
From: https://drive.google.com/uc?id=1ehxgfHTsMZXy6YzlFKGJOsBKQ5rrvMnd
To: C:\Users\sergei.rybakov\projects\notebooks\pancreas.h5ad
126MB [00:40, 3.14MB/s]
```

[5]:
```
'pancreas.h5ad'
```

[6]:
```
adata_all = sc.read('pancreas.h5ad')
```

[7]:
```
adata = adata_all.raw.to_adata()
adata = remove_sparsity(adata)
```

(continues on next page)

(continued from previous page)

```
source_adata = adata[~adata.obs[condition_key].isin(target_conditions)].copy()
target_adata = adata[adata.obs[condition_key].isin(target_conditions)].copy()
```

[8]:
```
source_adata
```

[8]:
```
AnnData object with n_obs × n_vars = 10294 × 1000
    obs: 'batch', 'study', 'cell_type', 'size_factors'
```

[9]:
```
target_adata
```

[9]:
```
AnnData object with n_obs × n_vars = 5387 × 1000
    obs: 'batch', 'study', 'cell_type', 'size_factors'
```

5.7.3 Create SCVI model and train it on reference dataset

[10]:
```
sca.dataset.setup_anndata(source_adata, batch_key=condition_key)
```

```
INFO      Using batches from adata.obs["study"]
INFO      No label_key inputted, assuming all cells have same label
INFO      Using data from adata.X
INFO      Computing library size prior per batch
INFO      Successfully registered anndata object containing 10294 cells, 1000 vars, 3
          batches, 1 labels, and 0 proteins. Also registered 0 extra categorical␣
↪covariates
          and 0 extra continuous covariates.
INFO      Please do not further modify adata until model is trained.
```

The parameters chosen here proofed to work best in the case of surgery with SCVI.

[12]:
```
vae = sca.models.SCVI(
    source_adata,
    n_layers=2,
    encode_covariates=True,
    deeply_inject_covariates=False,
    use_layer_norm="both",
    use_batch_norm="none",
)
```

[13]:
```
vae.train(n_epochs=vae_epochs, frequency=1, early_stopping_kwargs=early_stopping_
↪kwargs)
```

```
INFO      Training for 500 epochs
INFO      KL warmup for 400 epochs
Training...:  19%|                                                    | 97/500 [03:
↪20<18:24,  2.74s/it]INFO      Reducing LR on epoch 97.
Training...:  24%|                                                   | 121/500 [04:26
↪<17:22,  2.75s/it]INFO      Reducing LR on epoch 121.
Training...:  29%|                                                  | 147/500 [05:38<16:
↪11,  2.75s/it]INFO      Reducing LR on epoch 147.
Training...:  31%|                                                  | 156/500 [06:02<15:
↪47,  2.76s/it]INFO      Reducing LR on epoch 156.
Training...:  33%|                                                 | 167/500 [06:33<15:15,␣
↪ 2.75s/it]INFO      Reducing LR on epoch 167.
Training...:  34%|                                                 | 169/500 [06:38<15:09,␣
↪ 2.75s/it]INFO
```

(continues on next page)

(continued from previous page)

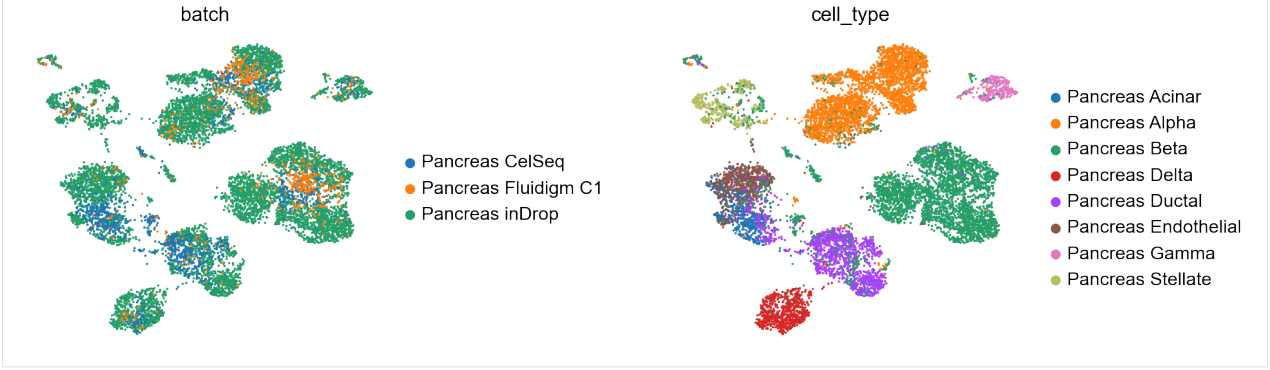
```
          Stopping early: no improvement of more than 0 nats in 10 epochs
INFO      If the early stopping criterion is too strong, please instantiate it with␣
↪different
          parameters in the train method.
Training...:  34%|                                                  | 169/500 [06:41<13:06,␣
↪ 2.37s/it]
INFO      Training is still in warming up phase. If your applications rely on the␣
↪posterior
          quality, consider training for more epochs or reducing the kl warmup.
INFO      Training time:  265 s. / 500 epochs
```

5.7.4 Create anndata file of latent representation and compute UMAP

[14]:
```
reference_latent = sc.AnnData(vae.get_latent_representation())
reference_latent.obs["cell_type"] = source_adata.obs[cell_type_key].tolist()
reference_latent.obs["batch"] = source_adata.obs[condition_key].tolist()
```

[15]:
```
sc.pp.neighbors(reference_latent, n_neighbors=8)
sc.tl.leiden(reference_latent)
sc.tl.umap(reference_latent)
sc.pl.umap(reference_latent,
           color=['batch', 'cell_type'],
           frameon=False,
           wspace=0.6,
           )
```

```
... storing 'cell_type' as categorical
... storing 'batch' as categorical
```



After pretraining the model can be saved for later use

[16]:
```
ref_path = 'ref_model/'
vae.save(ref_path, overwrite=True)
```

5.7.5 Perform surgery on reference model and train on query dataset

[17]:
```
model = sca.models.SCVI.load_query_data(
    target_adata,
    ref_path,
    freeze_dropout = True,
)
```

```
INFO      .obs[_scvi_labels] not found in target, assuming every cell is same category
INFO      Using data from adata.X
INFO      Computing library size prior per batch
INFO      Registered keys:['X', 'batch_indices', 'local_l_mean', 'local_l_var',
↪'labels']
INFO      Successfully registered anndata object containing 5387 cells, 1000 vars, 5␣
↪batches,
          1 labels, and 0 proteins. Also registered 0 extra categorical covariates␣
↪and 0
          extra continuous covariates.
WARNING   Make sure the registered X field in anndata contains unnormalized count␣
↪data.
```

[18]:
```
model.train(n_epochs=surgery_epochs, frequency=1, early_stopping_kwargs=early_
↪stopping_kwargs, weight_decay=0)
```

```
INFO      Training for 500 epochs
INFO      KL warmup for 400 epochs
Training...:  11%|                                                                | 55/
↪500 [01:07<09:02,  1.22s/it]INFO      Reducing LR on epoch 55.
Training...:  14%|                                                               | 70/500␣
↪[01:25<08:42,  1.21s/it]INFO      Reducing LR on epoch 70.
Training...:  14%|                                                               | 72/500␣
↪[01:27<08:40,  1.22s/it]INFO
          Stopping early: no improvement of more than 0 nats in 10 epochs
INFO      If the early stopping criterion is too strong, please instantiate it with␣
↪different
          parameters in the train method.
Training...:  14%|                                                               | 72/500␣
↪[01:28<08:48,  1.24s/it]
INFO      Training is still in warming up phase. If your applications rely on the␣
↪posterior
          quality, consider training for more epochs or reducing the kl warmup.
INFO      Training time:  53 s. / 500 epochs
```

[19]:
```
query_latent = sc.AnnData(model.get_latent_representation())
query_latent.obs['cell_type'] = target_adata.obs[cell_type_key].tolist()
query_latent.obs['batch'] = target_adata.obs[condition_key].tolist()
```

```
WARNING   Make sure the registered X field in anndata contains unnormalized count␣
↪data.
```

[20]:
```
sc.pp.neighbors(query_latent)
sc.tl.leiden(query_latent)
sc.tl.umap(query_latent)
plt.figure()
sc.pl.umap(
    query_latent,
    color=["batch", "cell_type"],
    frameon=False,
```

(continues on next page)

(continued from previous page)

```
    wspace=0.6,
)
```

```
... storing 'cell_type' as categorical
... storing 'batch' as categorical
```

```
<Figure size 320x320 with 0 Axes>
```

[21]:
```
surgery_path = 'surgery_model'
model.save(surgery_path, overwrite=True)
```

5.7.6 Get latent representation of reference + query dataset and compute UMAP

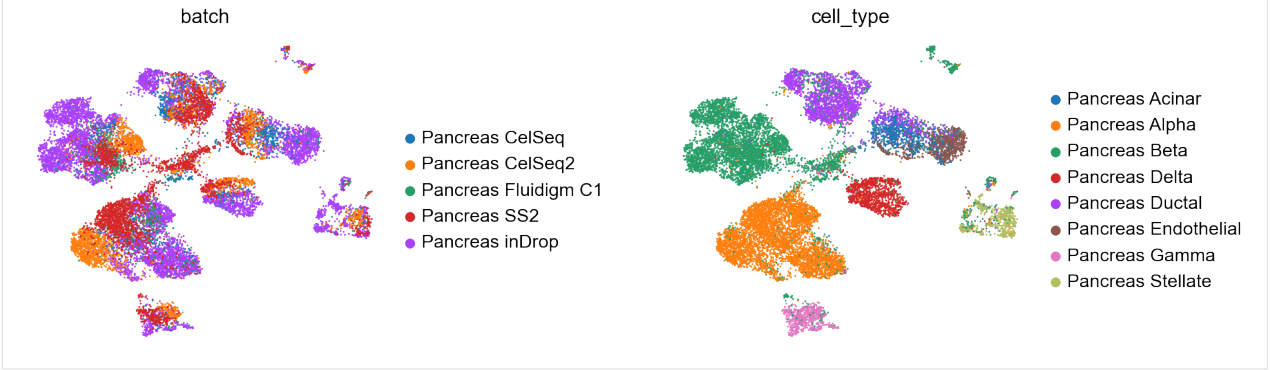
[22]:
```
adata_full = source_adata.concatenate(target_adata)
full_latent = sc.AnnData(model.get_latent_representation(adata=adata_full))
full_latent.obs['cell_type'] = adata_full.obs[cell_type_key].tolist()
full_latent.obs['batch'] = adata_full.obs[condition_key].tolist()
```

```
INFO      Input adata not setup with scvi. attempting to transfer anndata setup
INFO      Using data from adata.X
INFO      Computing library size prior per batch
INFO      Registered keys:['X', 'batch_indices', 'local_l_mean', 'local_l_var',
↪'labels']
INFO      Successfully registered anndata object containing 15681 cells, 1000 vars, 5
          batches, 1 labels, and 0 proteins. Also registered 0 extra categorical␣
↪covariates
          and 0 extra continuous covariates.
```

[23]:
```
sc.pp.neighbors(full_latent)
sc.tl.leiden(full_latent)
sc.tl.umap(full_latent)
plt.figure()
sc.pl.umap(
    full_latent,
    color=["batch", "cell_type"],
    frameon=False,
    wspace=0.6,
)
```

```
... storing 'cell_type' as categorical
... storing 'batch' as categorical
```

```
<Figure size 320x320 with 0 Axes>
```



5.8 Semi-supervised surgery pipeline with SCANVI

[1]:
```
import os
os.chdir('../')
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
warnings.simplefilter(action='ignore', category=UserWarning)
```

[2]:
```
import scanpy as sc
import torch
import scarchest as sca
from scarchest.dataset.trvae.data_handling import remove_sparsity
import matplotlib.pyplot as plt
import numpy as np
import gdown
```

[3]:
```
sc.settings.set_figure_params(dpi=200, frameon=False)
sc.set_figure_params(dpi=200)
sc.set_figure_params(figsize=(4, 4))
torch.set_printoptions(precision=3, sci_mode=False, edgeitems=7)
```

5.8.1 Set relevant anndata.obs labels and training length

Here we use the CelSeq2 and SS2 studies as query data and the other 3 studies as reference atlas. We strongly suggest to use earlystopping to avoid over-fitting. The best earlystopping criteria are the 'elbo' for SCVI pretraining and also for unlabelled surgery training and 'accurarcy' for semi-supervised SCANVI training.

[4]:
```
condition_key = 'study'
cell_type_key = 'cell_type'
target_conditions = ['Pancreas CelSeq2', 'Pancreas SS2']

vae_epochs = 500
scanvi_epochs = 200
surgery_epochs = 500

early_stopping_kwargs = {
    "early_stopping_metric": "elbo",
```

(continues on next page)

(continued from previous page)

```
    "save_best_state_metric": "elbo",
    "patience": 10,
    "threshold": 0,
    "reduce_lr_on_plateau": True,
    "lr_patience": 8,
    "lr_factor": 0.1,
}
early_stopping_kwargs_scanvi = {
    "early_stopping_metric": "accuracy",
    "save_best_state_metric": "accuracy",
    "on": "full_dataset",
    "patience": 10,
    "threshold": 0.001,
    "reduce_lr_on_plateau": True,
    "lr_patience": 8,
    "lr_factor": 0.1,
}
early_stopping_kwargs_surgery = {
    "early_stopping_metric": "elbo",
    "save_best_state_metric": "elbo",
    "on": "full_dataset",
    "patience": 10,
    "threshold": 0.001,
    "reduce_lr_on_plateau": True,
    "lr_patience": 8,
    "lr_factor": 0.1,
}
```

5.8.2 Download Dataset and split into reference dataset and query dataset

[5]:
```
url = 'https://drive.google.com/uc?id=1ehxgfHTsMZXy6YzlFKGJOsBKQ5rrvMnd'
output = 'pancreas.h5ad'
gdown.download(url, output, quiet=False)
```

```
Downloading...
From: https://drive.google.com/uc?id=1ehxgfHTsMZXy6YzlFKGJOsBKQ5rrvMnd
To: C:\Users\sergei.rybakov\projects\notebooks\pancreas.h5ad
126MB [00:29, 4.31MB/s]
```

[5]:
```
'pancreas.h5ad'
```

[6]:
```
adata_all = sc.read('pancreas.h5ad')
```

[7]:
```
adata = adata_all.raw.to_adata()
adata = remove_sparsity(adata)
source_adata = adata[~adata.obs[condition_key].isin(target_conditions)].copy()
target_adata = adata[adata.obs[condition_key].isin(target_conditions)].copy()
```

[8]:
```
source_adata
```

[8]:
```
AnnData object with n_obs × n_vars = 10294 × 1000
    obs: 'batch', 'study', 'cell_type', 'size_factors'
```

[9]:
```
target_adata
```

[9]:
```
AnnData object with n_obs × n_vars = 5387 × 1000
    obs: 'batch', 'study', 'cell_type', 'size_factors'
```

5.8.3 Create SCANVI model and train it on fully labelled reference dataset

[10]:
```
sca.dataset.setup_anndata(source_adata, batch_key=condition_key, labels_key=cell_type_
↪key)
```

```
INFO      Using batches from adata.obs["study"]
INFO      Using labels from adata.obs["cell_type"]
INFO      Using data from adata.X
INFO      Computing library size prior per batch
INFO      Successfully registered anndata object containing 10294 cells, 1000 vars, 3
          batches, 8 labels, and 0 proteins. Also registered 0 extra categorical␣
↪covariates
          and 0 extra continuous covariates.
INFO      Please do not further modify adata until model is trained.
```

The parameters chosen here proofed to work best in the case of surgery with SCANVI.

[11]:
```
vae = sca.models.SCANVI(
    source_adata,
    "Unknown",
    n_layers=2,
    encode_covariates=True,
    deeply_inject_covariates=False,
    use_layer_norm="both",
    use_batch_norm="none",
)
```

[12]:
```
print("Labelled Indices: ", len(vae._labeled_indices))
print("Unlabelled Indices: ", len(vae._unlabeled_indices))
```

```
Labelled Indices:  10294
Unlabelled Indices:  0
```

[13]:
```
vae.train(
    n_epochs_unsupervised=vae_epochs,
    n_epochs_semisupervised=scanvi_epochs,
    unsupervised_trainer_kwargs=dict(early_stopping_kwargs=early_stopping_kwargs),
    semisupervised_trainer_kwargs=dict(metrics_to_monitor=["elbo", "accuracy"],
                                       early_stopping_kwargs=early_stopping_kwargs_
↪scanvi),
    frequency=1
)
```

```
INFO      Training Unsupervised Trainer for 500 epochs.
INFO      Training SemiSupervised Trainer for 200 epochs.
INFO      KL warmup for 400 epochs
Training...:  20%|                                                          | 99/500 [04:
↪03<18:22,  2.75s/it]INFO      Reducing LR on epoch 99.
Training...:  25%|                                                        | 125/500 [05:15
↪<17:12,  2.75s/it]INFO      Reducing LR on epoch 125.
Training...:  25%|                                                        | 127/500 [05:20
↪<17:06,  2.75s/it]INFO
          Stopping early: no improvement of more than 0 nats in 10 epochs
```

(continues on next page)

(continued from previous page)

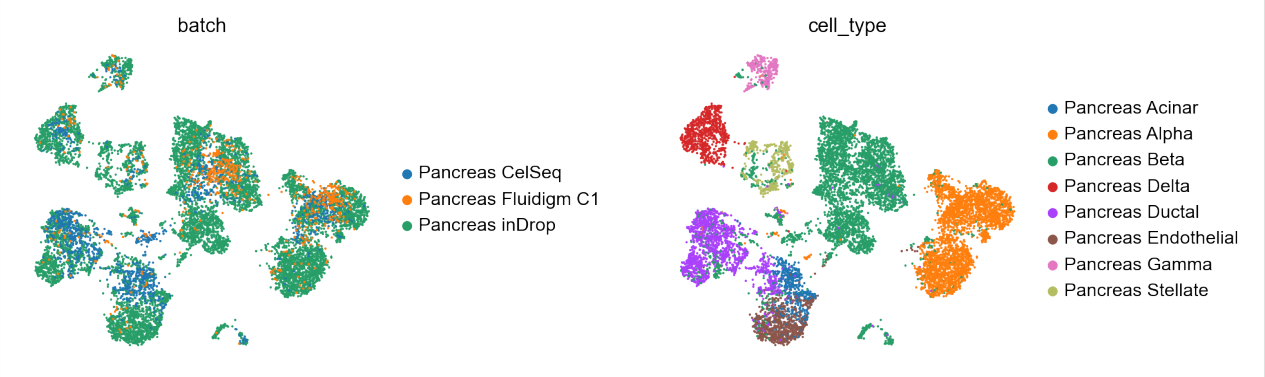
```
INFO      If the early stopping criterion is too strong, please instantiate it with␣
↪different
          parameters in the train method.
Training...:  25%|                                                    | 127/500 [05:23
↪<15:50,  2.55s/it]
INFO      Training is still in warming up phase. If your applications rely on the␣
↪posterior
          quality, consider training for more epochs or reducing the kl warmup.
INFO      Training time:  214 s. / 500 epochs
INFO      KL warmup phase exceeds overall training phaseIf your applications rely on␣
↪the
          posterior quality, consider training for more epochs or reducing the kl␣
↪warmup.
INFO      KL warmup for 400 epochs
Training...:  19%|                                                     | 38/200 [05:
↪51<25:02,  9.28s/it]INFO      Reducing LR on epoch 38.
Training...:  20%|                                                    | 40/200 [06:
↪10<24:43,  9.27s/it]INFO
          Stopping early: no improvement of more than 0.001 nats in 10 epochs
INFO      If the early stopping criterion is too strong, please instantiate it with␣
↪different
          parameters in the train method.
Training...:  20%|                                                    | 40/200 [06:
↪19<25:18,  9.49s/it]
INFO      Training is still in warming up phase. If your applications rely on the␣
↪posterior
          quality, consider training for more epochs or reducing the kl warmup.
INFO      Training time:  228 s. / 200 epochs
```

5.8.4 Create anndata file of latent representation and compute UMAP

[14]:
```
reference_latent = sc.AnnData(vae.get_latent_representation())
reference_latent.obs["cell_type"] = source_adata.obs[cell_type_key].tolist()
reference_latent.obs["batch"] = source_adata.obs[condition_key].tolist()
```

[15]:
```
sc.pp.neighbors(reference_latent, n_neighbors=8)
sc.tl.leiden(reference_latent)
sc.tl.umap(reference_latent)
sc.pl.umap(reference_latent,
           color=['batch', 'cell_type'],
           frameon=False,
           wspace=0.6,
           )
```

```
... storing 'cell_type' as categorical
... storing 'batch' as categorical
```



One can also compute the accuracy of the learned classifier

[16]:
```
reference_latent.obs['predictions'] = vae.predict()
print("Acc: {}".format(np.mean(reference_latent.obs.predictions == reference_latent.
↪obs.cell_type)))
```

```
Acc: 0.9619195647950263
```

After pretraining the model can be saved for later use

[17]:
```
ref_path = 'ref_model/'
vae.save(ref_path, overwrite=True)
```

5.8.5 Perform surgery on reference model and train on query dataset without cell type labels

[18]:
```
model = sca.models.SCANVI.load_query_data(
    target_adata,
    ref_path,
    freeze_dropout = True,
)
model._unlabeled_indices = np.arange(target_adata.n_obs)
model._labeled_indices = []
print("Labelled Indices: ", len(model._labeled_indices))
print("Unlabelled Indices: ", len(model._unlabeled_indices))
```

```
INFO      Using data from adata.X
INFO      Computing library size prior per batch
INFO      Registered keys:['X', 'batch_indices', 'local_l_mean', 'local_l_var',
↪'labels']
INFO      Successfully registered anndata object containing 5387 cells, 1000 vars, 5␣
↪batches,
          8 labels, and 0 proteins. Also registered 0 extra categorical covariates␣
↪and 0
          extra continuous covariates.
WARNING   Make sure the registered X field in anndata contains unnormalized count␣
↪data.
Labelled Indices:  0
Unlabelled Indices:  5387
```

[19]:
```
model.train(
    n_epochs_semisupervised=surgery_epochs,
```

(continues on next page)

(continued from previous page)

```
    train_base_model=False,
    semisupervised_trainer_kwargs=dict(metrics_to_monitor=["accuracy", "elbo"],
                                       weight_decay=0,
                                       early_stopping_kwargs=early_stopping_kwargs_
↪surgery
                                      ),
    frequency=1
)
```

```
INFO      Training Unsupervised Trainer for 400 epochs.
INFO      Training SemiSupervised Trainer for 500 epochs.
INFO      KL warmup for 400 epochs
Training...:  22%|                                                          | 111/500 [07:08
↪<25:03,  3.87s/it]INFO      Reducing LR on epoch 111.
Training...:  23%|                                                         | 113/500 [07:16
↪<24:55,  3.86s/it]INFO
          Stopping early: no improvement of more than 0.001 nats in 10 epochs
INFO      If the early stopping criterion is too strong, please instantiate it with␣
↪different
          parameters in the train method.
Training...:  23%|                                                         | 113/500 [07:20
↪<25:08,  3.90s/it]
INFO      Training is still in warming up phase. If your applications rely on the␣
↪posterior
          quality, consider training for more epochs or reducing the kl warmup.
INFO      Training time:  217 s. / 500 epochs
```

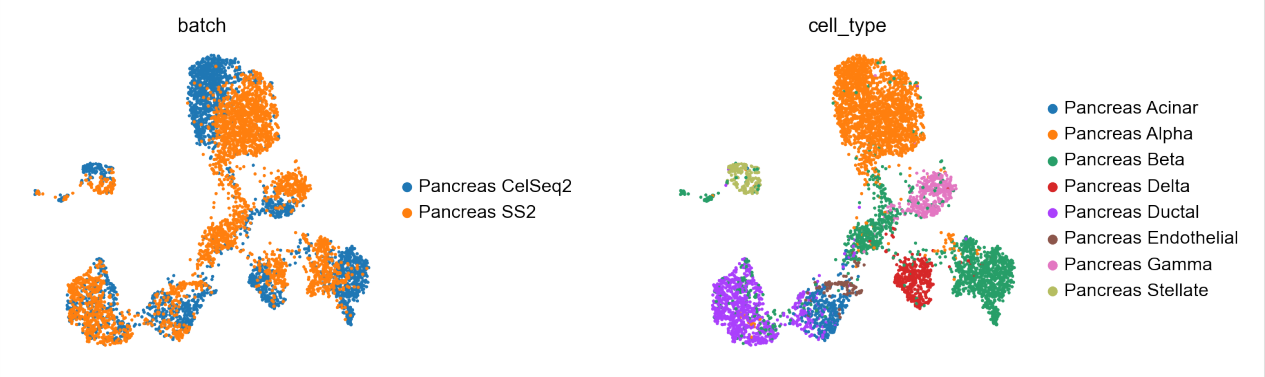
[20]:
```
query_latent = sc.AnnData(model.get_latent_representation())
query_latent.obs['cell_type'] = target_adata.obs[cell_type_key].tolist()
query_latent.obs['batch'] = target_adata.obs[condition_key].tolist()
```

```
WARNING   Make sure the registered X field in anndata contains unnormalized count␣
↪data.
```

[21]:
```
sc.pp.neighbors(query_latent)
sc.tl.leiden(query_latent)
sc.tl.umap(query_latent)
plt.figure()
sc.pl.umap(
    query_latent,
    color=["batch", "cell_type"],
    frameon=False,
    wspace=0.6,
)
```

```
... storing 'cell_type' as categorical
... storing 'batch' as categorical
```

```
<Figure size 320x320 with 0 Axes>
```



[22]:
```
surgery_path = 'surgery_model'
model.save(surgery_path, overwrite=True)
```

5.8.6 Compute Accuracy of model classifier for query dataset and compare predicted and observed cell types

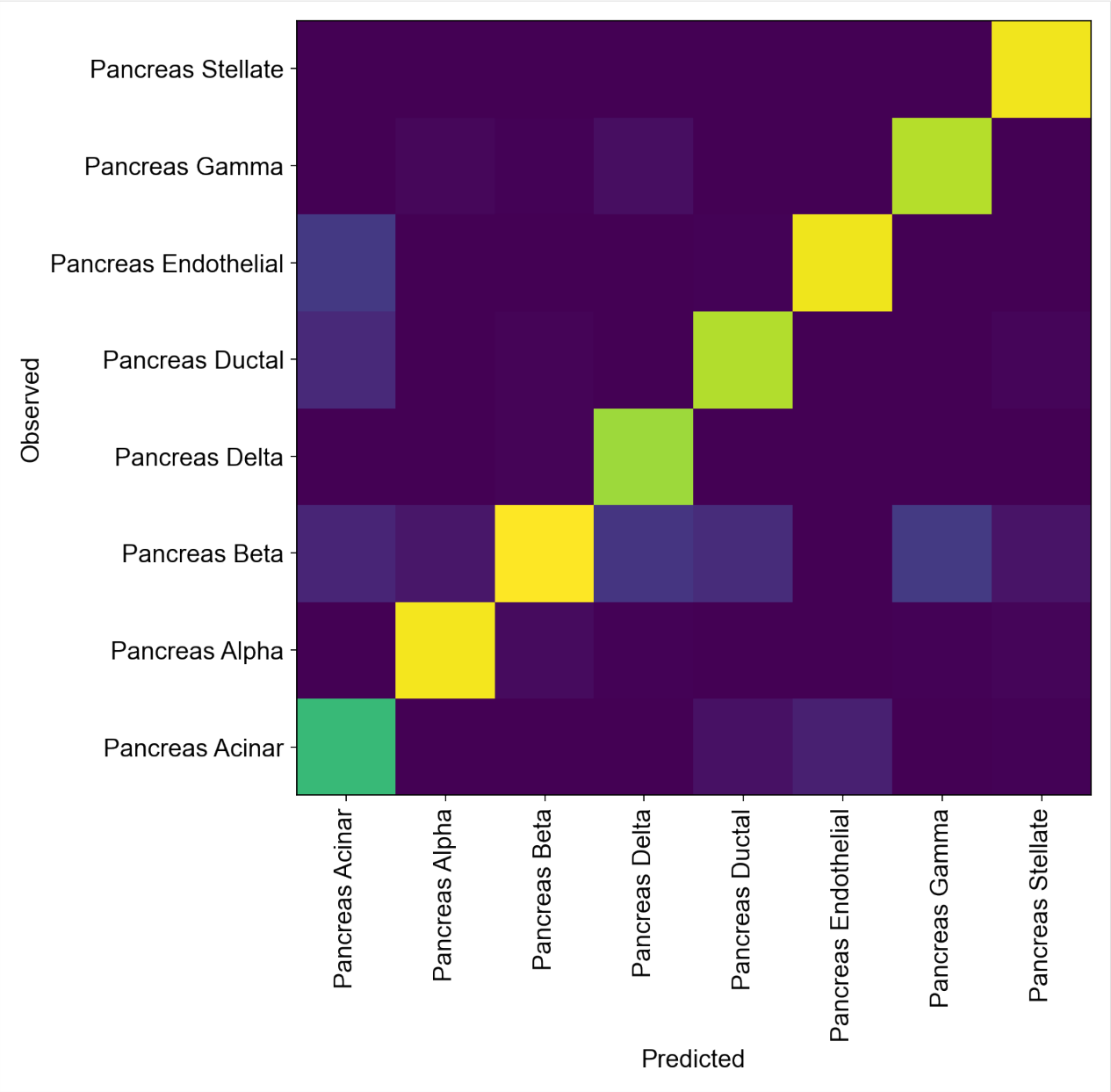
[23]:
```
query_latent.obs['predictions'] = model.predict()
print("Acc: {}".format(np.mean(query_latent.obs.predictions == query_latent.obs.cell_
↪type)))
```

```
WARNING  Make sure the registered X field in anndata contains unnormalized count␣
↪data.
Acc: 0.8791535177278633
```

[24]:
```
df = query_latent.obs.groupby(["cell_type", "predictions"]).size().unstack(fill_
↪value=0)
norm_df = df / df.sum(axis=0)

plt.figure(figsize=(8, 8))
_ = plt.pcolor(norm_df)
_ = plt.xticks(np.arange(0.5, len(df.columns), 1), df.columns, rotation=90)
_ = plt.yticks(np.arange(0.5, len(df.index), 1), df.index)
plt.xlabel("Predicted")
plt.ylabel("Observed")
```

[24]:
```
Text(0, 0.5, 'Observed')
```



5.8.7 Get latent representation of reference + query dataset and compute UMAP

[25]:
```
adata_full = source_adata.concatenate(target_adata)
full_latent = sc.AnnData(model.get_latent_representation(adata=adata_full))
full_latent.obs['cell_type'] = adata_full.obs[cell_type_key].tolist()
full_latent.obs['batch'] = adata_full.obs[condition_key].tolist()
```

```
INFO      Input adata not setup with scvi. attempting to transfer anndata setup
INFO      Using data from adata.X
INFO      Computing library size prior per batch
INFO      Registered keys:['X', 'batch_indices', 'local_l_mean', 'local_l_var',
↪'labels']
INFO      Successfully registered anndata object containing 15681 cells, 1000 vars, 5
```

(continues on next page)

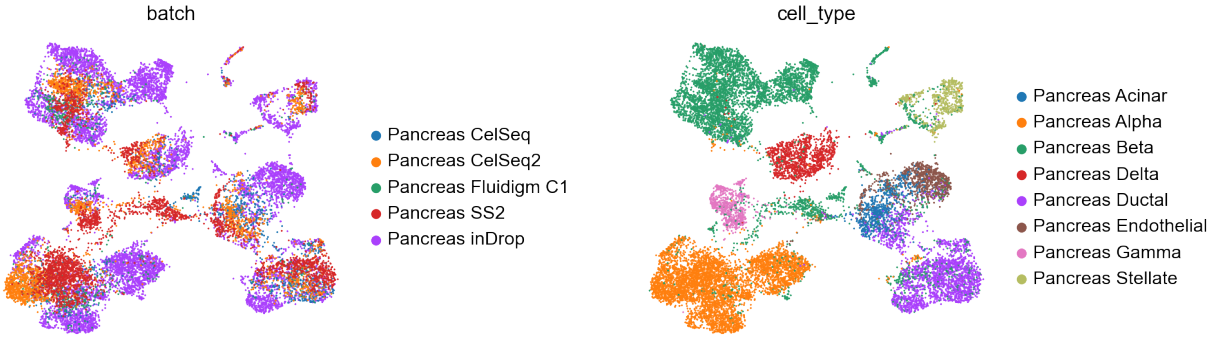
(continued from previous page)

```
        batches, 8 labels, and 0 proteins. Also registered 0 extra categorical␣
↪covariates
        and 0 extra continuous covariates.
```

[26]:
```
sc.pp.neighbors(full_latent)
sc.tl.leiden(full_latent)
sc.tl.umap(full_latent)
plt.figure()
sc.pl.umap(
    full_latent,
    color=["batch", "cell_type"],
    frameon=False,
    wspace=0.6,
)
```

```
... storing 'cell_type' as categorical
... storing 'batch' as categorical
```

```
<Figure size 320x320 with 0 Axes>
```



5.8.8 Comparison of observed and predicted celltypes for reference + query dataset

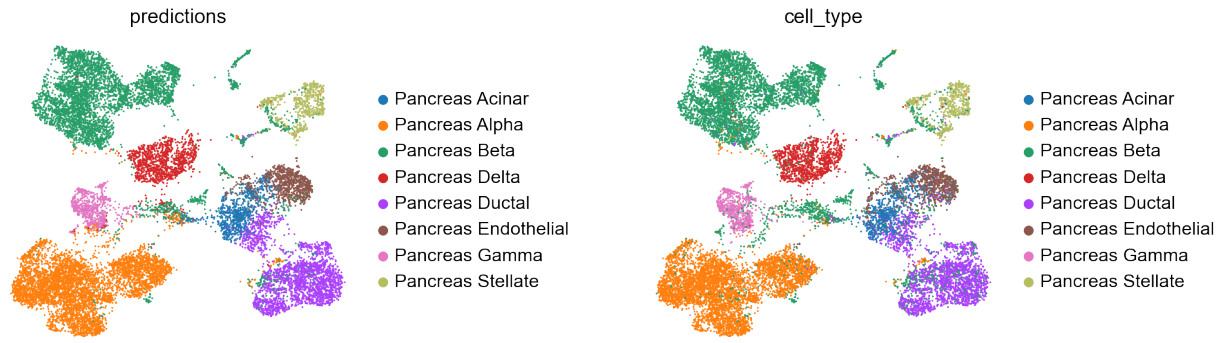
[27]:
```
full_latent.obs['predictions'] = model.predict(adata=adata_full)
print("Acc: {}".format(np.mean(full_latent.obs.predictions == full_latent.obs.cell_
↪type)))
```

```
Acc: 0.933486384796888
```

[28]:
```
sc.pp.neighbors(full_latent)
sc.tl.leiden(full_latent)
sc.tl.umap(full_latent)
plt.figure()
sc.pl.umap(
    full_latent,
    color=["predictions", "cell_type"],
    frameon=False,
    wspace=0.6,
)
```

```
... storing 'predictions' as categorical
```

```
<Figure size 320x320 with 0 Axes>
```



5.9 Multi-Modal Surgery Pipeline with TOTALVI

[1]:
```
import os
os.chdir('../')
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
warnings.simplefilter(action='ignore', category=UserWarning)
```

[2]:
```
import scanpy as sc
import anndata
import torch
import scarchest as sca
import matplotlib.pyplot as plt
import numpy as np
import scvi as scv
import pandas as pd
```

[3]:
```
sc.settings.set_figure_params(dpi=200, frameon=False)
sc.set_figure_params(dpi=200)
sc.set_figure_params(figsize=(4, 4))
torch.set_printoptions(precision=3, sci_mode=False, edgeitems=7)
```

5.9.1 Data loading and preprocessing

For totalVI, we will treat two CITE-seq PBMC datasets from 10X Genomics as the reference. These datasets were already filtered for outliers like doublets, as described in the totalVI manuscript. There are 14 proteins in the reference.

[4]:
```
adata_ref = scv.data.pbmcs_10x_cite_seq(run_setup_anndata=False)
```

```
INFO      Downloading file at data/pbmc_10k_protein_v3.h5ad
Downloading...: 24938it [00:04, 5776.87it/s]
INFO      Downloading file at data/pbmc_5k_protein_v3.h5ad
Downloading...: 100%|| 18295/18295.0 [00:03<00:00, 5772.53it/s]
```

```
Observation names are not unique. To make them unique, call `.obs_names_make_unique`.
```

[5]:
```
adata_query = scv.data.dataset_10x("pbmc_10k_v3")
adata_query.obs["batch"] = "PBMC 10k (RNA only)"
# put matrix of zeros for protein expression (considered missing)
pro_exp = adata_ref.obsm["protein_expression"]
data = np.zeros((adata_query.n_obs, pro_exp.shape[1]))
adata_query.obsm["protein_expression"] = pd.DataFrame(columns=pro_exp.columns,␣
↪index=adata_query.obs_names, data = data)
```

```
INFO     Downloading file at data/10X\pbmc_10k_v3\filtered_feature_bc_matrix.h5
Downloading...: 37492it [00:08, 4267.50it/s]
```

```
Variable names are not unique. To make them unique, call `.var_names_make_unique`.
Variable names are not unique. To make them unique, call `.var_names_make_unique`.
```

Now to concatenate the objects, which intersects the genes properly.

[6]:
```
adata_full = anndata.concat([adata_ref, adata_query])
```

```
Observation names are not unique. To make them unique, call `.obs_names_make_unique`.
```

And split them back up into reference and query (but now genes are properly aligned between objects).

[7]:
```
adata_ref = adata_full[np.logical_or(adata_full.obs.batch == "PBMC5k", adata_full.obs.
↪batch == "PBMC10k")].copy()
adata_query = adata_full[adata_full.obs.batch == "PBMC 10k (RNA only)"].copy()
```

```
Observation names are not unique. To make them unique, call `.obs_names_make_unique`.
```

We run gene selection on the reference, because that's all that will be avaialble to us at first.

[8]:
```
sc.pp.highly_variable_genes(
    adata_ref,
    n_top_genes=4000,
    flavor="seurat_v3",
    batch_key="batch",
    subset=True,
)
```

```
Observation names are not unique. To make them unique, call `.obs_names_make_unique`.
Observation names are not unique. To make them unique, call `.obs_names_make_unique`.
```

Finally, we use these selected genes for the query dataset as well.

[9]:
```
adata_query = adata_query[:, adata_ref.var_names].copy()
```

5.9.2 Create TOTALVI model and train it on CITE-seq reference dataset

[10]:
```
sca.dataset.setup_anndata(
    adata_ref,
    batch_key="batch",
    protein_expression_obsm_key="protein_expression"
)
```

```
INFO     Using batches from adata.obs["batch"]
INFO     No label_key inputted, assuming all cells have same label
INFO     Using data from adata.X
INFO     Computing library size prior per batch
INFO     Using protein expression from adata.obsm['protein_expression']
```

(continues on next page)

(continued from previous page)

```
INFO      Using protein names from columns of adata.obsm['protein_expression']
INFO      Successfully registered anndata object containing 10849 cells, 4000 vars, 2
          batches, 1 labels, and 14 proteins. Also registered 0 extra categorical␣
↪covariates
          and 0 extra continuous covariates.
INFO      Please do not further modify adata until model is trained.
```

[11]:
```
arches_params = dict(
    use_layer_norm="both",
    use_batch_norm="none",
)
vae_ref = sca.models.TOTALVI(
    adata_ref,
    use_cuda=True,
    **arches_params
)
```

[12]:
```
vae_ref.train()
```

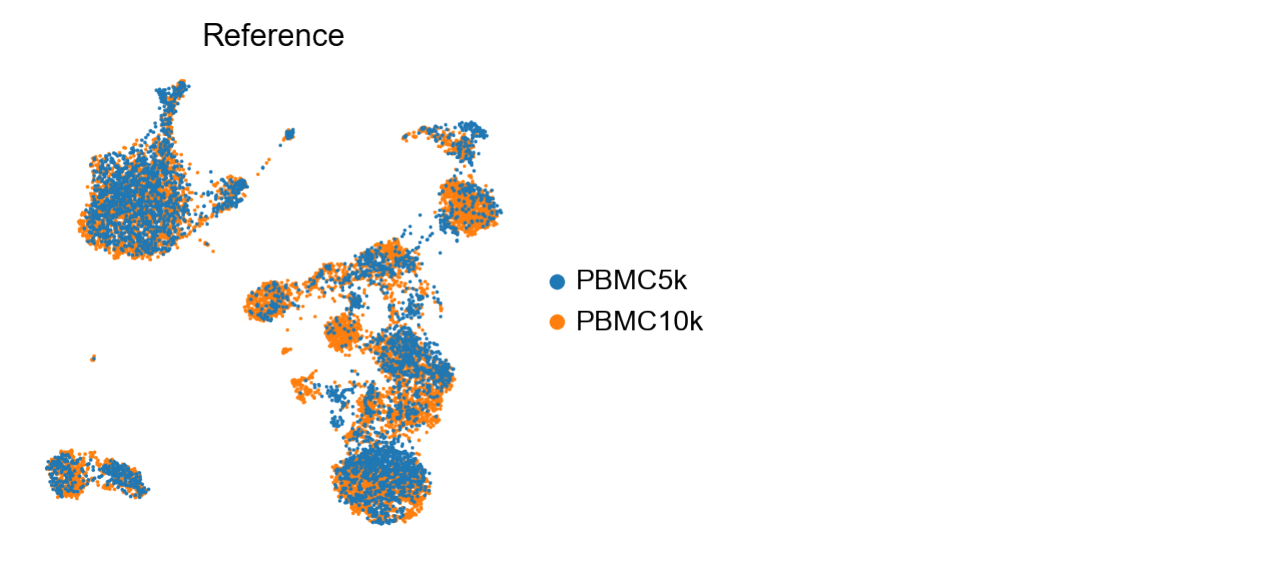
```
INFO      Training for 400 epochs.
INFO      KL warmup for 8136.75 iterations
Training...:  67%|                       | 267/400 [18:30<06:15,  2.83s/it]INFO     ␣
↪Reducing LR on epoch 267.
Training...:  76%|                | 304/400 [20:05<03:48,  2.38s/it]INFO     ␣
↪Reducing LR on epoch 304.
Training...:  96%|   | 382/400 [23:37<00:48,  2.69s/it]INFO      Reducing LR on epoch␣
↪382.
Training...: 100%|| 400/400 [24:21<00:00,  3.65s/it]
INFO      Training time:  1376 s. / 400 epochs
```

5.9.3 Save Latent representation and visualize RNA data

[13]:
```
adata_ref.obsm["X_totalVI"] = vae_ref.get_latent_representation()
sc.pp.neighbors(adata_ref, use_rep="X_totalVI")
sc.tl.umap(adata_ref, min_dist=0.4)
```

[14]:
```
sc.pl.umap(
    adata_ref,
    color=["batch"],
    frameon=False,
    ncols=1,
    title="Reference"
)
```

```
... storing 'batch' as categorical
```



5.9.4 Save trained reference model

[15]:
```
dir_path = "saved_model/"
vae_ref.save(dir_path, overwrite=True)
```

5.9.5 Perform surgery on reference model and train on query dataset without protein data

[16]:
```
vae_q = sca.models.TOTALVI.load_query_data(
    adata_query,
    dir_path,
    freeze_expression=True
)
```

```
INFO      .obs[_scvi_labels] not found in target, assuming every cell is same category
INFO      Found batches with missing protein expression
INFO      Using data from adata.X
INFO      Computing library size prior per batch
INFO      Registered keys:['X', 'batch_indices', 'local_l_mean', 'local_l_var',
↪'labels',
          'protein_expression']
INFO      Successfully registered anndata object containing 11769 cells, 4000 vars, 3
          batches, 1 labels, and 14 proteins. Also registered 0 extra categorical␣
↪covariates
          and 0 extra continuous covariates.
```

[17]:
```
vae_q.train(200, weight_decay=0.0)
```

```
INFO      Training for 200 epochs.
INFO      KL warmup for 8826.75 iterations
Training...:  68%|                    | 136/200 [11:10<04:09,  3.90s/it]INFO      ␣
↪Reducing LR on epoch 136.
Training...:  76%|                | 151/200 [12:03<03:00,  3.69s/it]INFO
```

(continues on next page)

(continued from previous page)

```
         Stopping early: no improvement of more than 0 nats in 45 epochs
INFO     If the early stopping criterion is too strong, please instantiate it with␣
↪different
         parameters in the train method.
Training...:  76%|                | 151/200 [12:06<03:55,  4.81s/it]
INFO     Training is still in warming up phase. If your applications rely on the␣
↪posterior
         quality, consider training for more epochs or reducing the kl warmup.
INFO     Training time:  689 s. / 200 epochs
```

[18]:
```
adata_query.obsm["X_totalVI"] = vae_q.get_latent_representation()
sc.pp.neighbors(adata_query, use_rep="X_totalVI")
sc.tl.umap(adata_query, min_dist=0.4)
```

5.9.6 Impute protein data for the query dataset and visualize

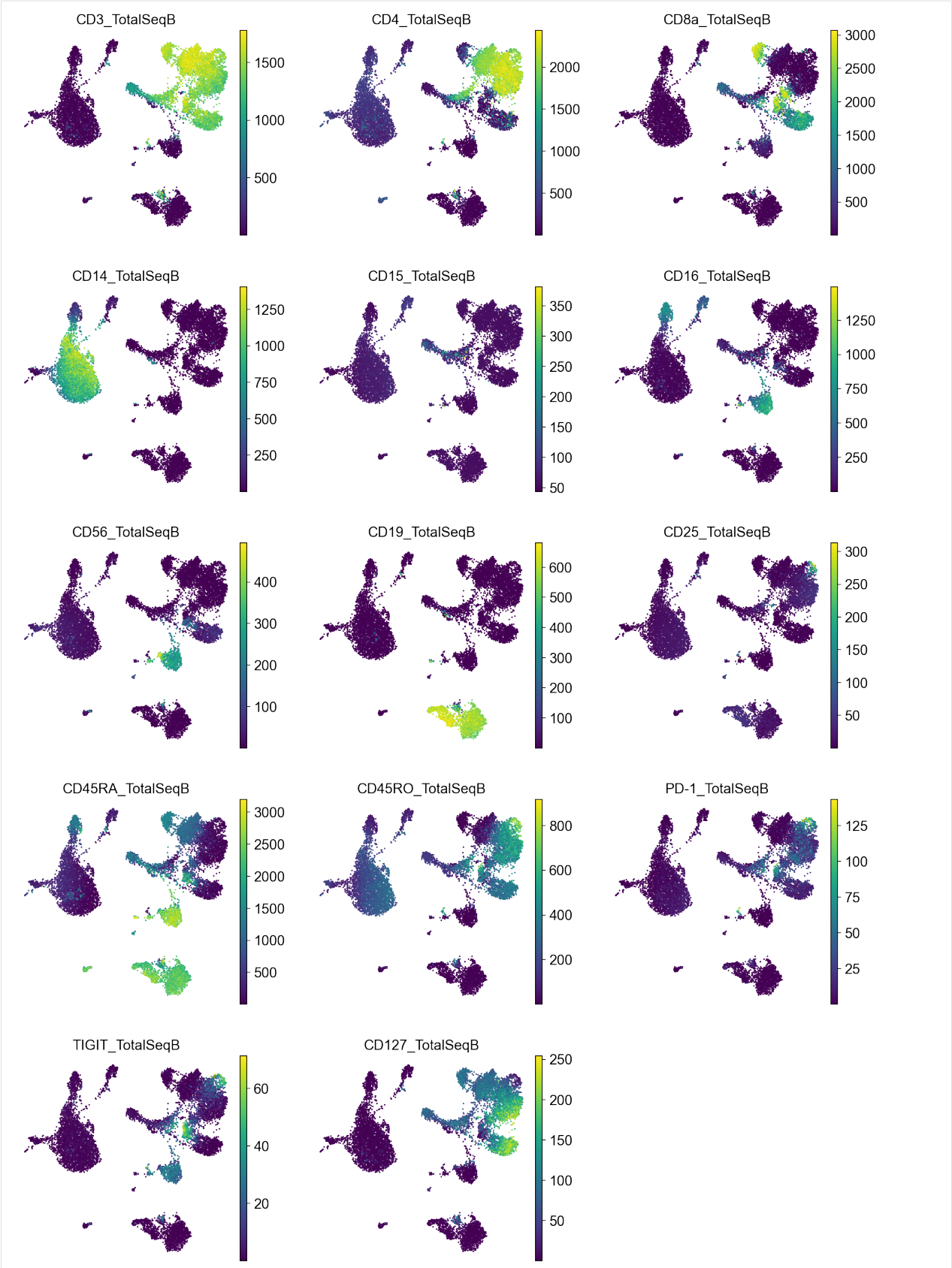
Impute the proteins that were observed in the reference, using the `transform_batch` parameter.

[19]:
```
_, imputed_proteins = vae_q.get_normalized_expression(
    adata_query,
    n_samples=25,
    return_mean=True,
    transform_batch=["PBMC10k", "PBMC5k"],
)
```

[20]:
```
adata_query.obs = pd.concat([adata_query.obs, imputed_proteins], axis=1)

sc.pl.umap(
    adata_query,
    color=imputed_proteins.columns,
    frameon=False,
    ncols=3,
)
```

```
... storing 'batch' as categorical
```


CD3_TotalSeqB
1500
1000
500
CD4_TotalSeqB
2000
1500
1000
500
CD8a_TotalSeqB
3000
2500
2000
1500
1000
500
CD14_TotalSeqB
1250
1000
750
500
250
CD15_TotalSeqB
350
300
250
200
150
100
50
CD16_TotalSeqB
1250
1000
750
500
250
CD56_TotalSeqB
400
300
200
100
CD19_TotalSeqB
600
500
400
300
200
100
CD25_TotalSeqB
300
250
200
150
100
50
CD45RA_TotalSeqB
3000
2500
2000
1500
1000
500
CD45RO_TotalSeqB
800
600
400
200
PD-1_TotalSeqB
125
100
75
50
25
TIGIT_TotalSeqB
60
40
20
CD127_TotalSeqB
250
200
150
100
50

5.9.7 Get latent representation of reference + query dataset and compute UMAP

[21]:
```
adata_full_new = adata_query.concatenate(adata_ref, batch_key="none")
```

```
Observation names are not unique. To make them unique, call `.obs_names_make_unique`.
Observation names are not unique. To make them unique, call `.obs_names_make_unique`.
Observation names are not unique. To make them unique, call `.obs_names_make_unique`.
```

[22]:
```
adata_full_new.obsm["X_totalVI"] = vae_q.get_latent_representation(adata_full_new)
sc.pp.neighbors(adata_full_new, use_rep="X_totalVI")
sc.tl.umap(adata_full_new, min_dist=0.3)
```

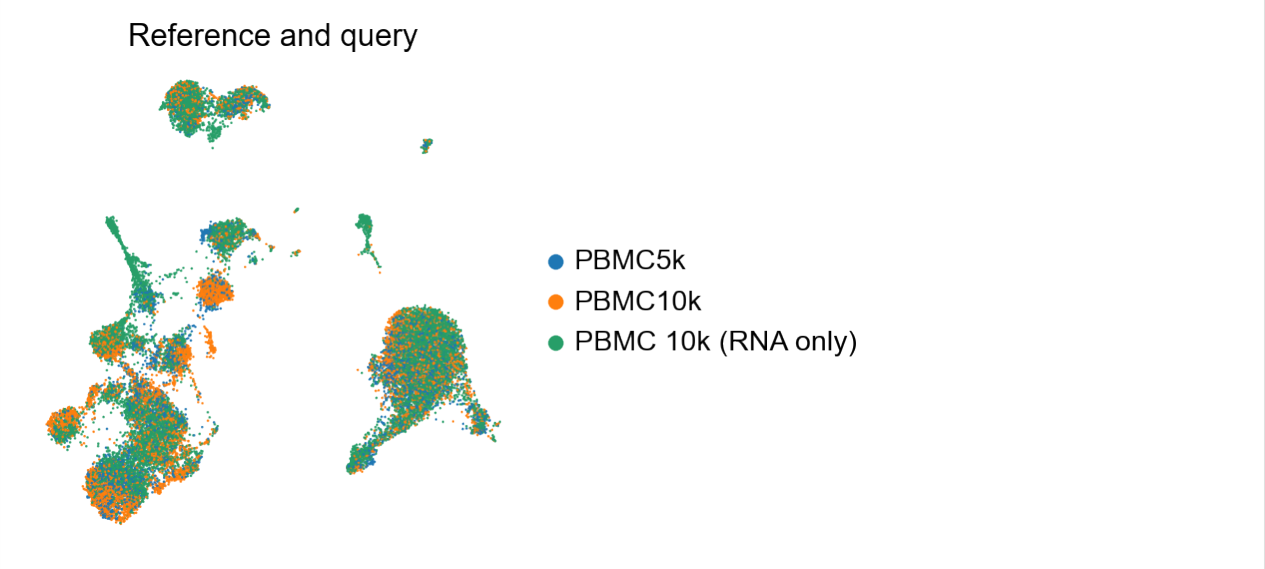
```
INFO      Input adata not setup with scvi. attempting to transfer anndata setup
INFO      Found batches with missing protein expression
INFO      Using data from adata.X
INFO      Computing library size prior per batch
INFO      Registered keys:['X', 'batch_indices', 'local_l_mean', 'local_l_var',
↪'labels',
          'protein_expression']
INFO      Successfully registered anndata object containing 22618 cells, 4000 vars, 3
          batches, 1 labels, and 14 proteins. Also registered 0 extra categorical␣
↪covariates
          and 0 extra continuous covariates.
```

[23]:
```
_, imputed_proteins_all = vae_q.get_normalized_expression(
    adata_full_new,
    n_samples=25,
    return_mean=True,
    transform_batch=["PBMC10k", "PBMC5k"],
)

for i, p in enumerate(imputed_proteins_all.columns):
    adata_full_new.obs[p] = imputed_proteins_all[p].to_numpy().copy()
```

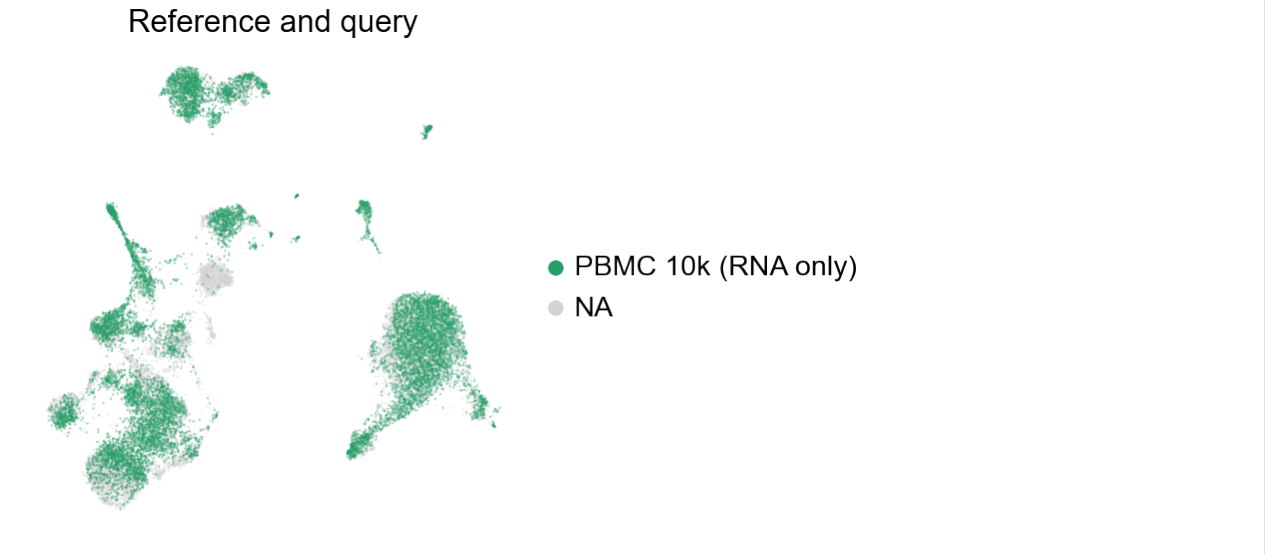
[24]:
```
perm_inds = np.random.permutation(np.arange(adata_full_new.n_obs))
sc.pl.umap(
    adata_full_new[perm_inds],
    color=["batch"],
    frameon=False,
    ncols=1,
    title="Reference and query"
)
```

```
C:\Users\sergei.rybakov\Apps\Miniconda3\envs\work\lib\site-packages\anndata\_core\
↪anndata.py:1213: ImplicitModificationWarning: Initializing view as actual.
  "Initializing view as actual.", ImplicitModificationWarning
Trying to set attribute `.obs` of view, copying.
Observation names are not unique. To make them unique, call `.obs_names_make_unique`.
Observation names are not unique. To make them unique, call `.obs_names_make_unique`.
... storing 'batch' as categorical
```



[25]:
```
ax = sc.pl.umap(
    adata_full_new,
    color="batch",
    groups=["PBMC 10k (RNA only)"],
    frameon=False,
    ncols=1,
    title="Reference and query",
    alpha=0.4
)
```

```
... storing 'batch' as categorical
```

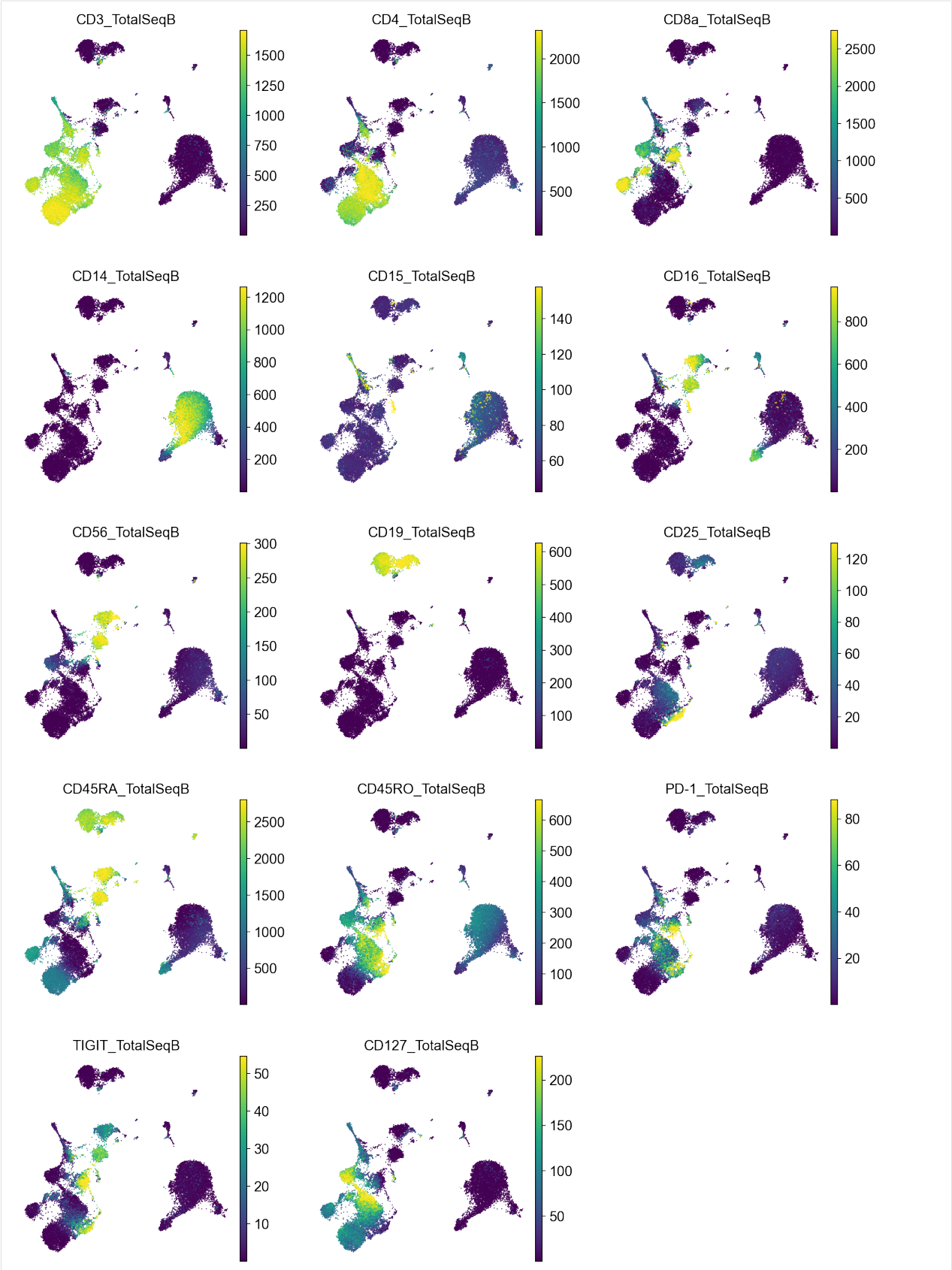


[26]:
```
sc.pl.umap(
    adata_full_new,
    color=imputed_proteins_all.columns,
    frameon=False,
    ncols=3,
    vmax="p99"
```

(continues on next page)

(continued from previous page)

```
)
```


CD3_TotalSeqB
1500
1250
1000
750
500
250
CD4_TotalSeqB
2000
1500
1000
500
CD8a_TotalSeqB
2500
2000
1500
1000
500
CD14_TotalSeqB
1200
1000
800
600
400
200
CD15_TotalSeqB
140
120
100
80
60
CD16_TotalSeqB
800
600
400
200
CD56_TotalSeqB
300
250
200
150
100
50
CD19_TotalSeqB
600
500
400
300
200
100
CD25_TotalSeqB
120
100
80
60
40
20
CD45RA_TotalSeqB
2500
2000
1500
1000
500
CD45RO_TotalSeqB
600
500
400
300
200
100
PD-1_TotalSeqB
80
60
40
20
TIGIT_TotalSeqB
50
40
30
20
10
CD127_TotalSeqB
200
150
100
50

5.10 Unsupervised surgery pipeline with TRVAE

[1]:
```
import os
os.chdir('../')
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
warnings.simplefilter(action='ignore', category=UserWarning)
```

[2]:
```
import scanpy as sc
import torch
import scarchest as sca
from scarchest.dataset.trvae.data_handling import remove_sparsity
import matplotlib.pyplot as plt
import numpy as np
import gdown
```

[3]:
```
sc.settings.set_figure_params(dpi=200, frameon=False)
sc.set_figure_params(dpi=200)
sc.set_figure_params(figsize=(4, 4))
torch.set_printoptions(precision=3, sci_mode=False, edgeitems=7)
```

5.10.1 Set relevant anndata.obs labels and training length

Here we use the CelSeq2 and SS2 studies as query data and the other 3 studies as reference atlas. We strongly suggest to use earlystopping to avoid over-fitting. The best earlystopping criteria is the 'val_unweighted_loss' for TRVAE.

[5]:
```
condition_key = 'study'
cell_type_key = 'cell_type'
target_conditions = ['Pancreas CelSeq2', 'Pancreas SS2']


trvae_epochs = 500
surgery_epochs = 500

early_stopping_kwargs = {
    "early_stopping_metric": "val_unweighted_loss",
    "threshold": 0,
    "patience": 20,
    "reduce_lr": True,
    "lr_patience": 13,
    "lr_factor": 0.1,
}
```

5.10.2 Download Dataset and split into reference dataset and query dataset

[6]:
```
url = 'https://drive.google.com/uc?id=1ehxgfHTsMZXy6YzlFKGJOsBKQ5rrvMnd'
output = 'pancreas.h5ad'
gdown.download(url, output, quiet=False)
```

```
Downloading...
From: https://drive.google.com/uc?id=1ehxgfHTsMZXy6YzlFKGJOsBKQ5rrvMnd
To: C:\Users\sergei.rybakov\projects\notebooks\pancreas.h5ad
126MB [00:35, 3.52MB/s]
```

[6]:
```
'pancreas.h5ad'
```

[7]:
```
adata_all = sc.read('pancreas.h5ad')
```

[8]:
```
adata = adata_all.raw.to_adata()
adata = remove_sparsity(adata)
source_adata = adata[~adata.obs[condition_key].isin(target_conditions)]
target_adata = adata[adata.obs[condition_key].isin(target_conditions)]
source_conditions = source_adata.obs[condition_key].unique().tolist()
```

[9]:
```
source_adata
```

[9]:
```
View of AnnData object with n_obs × n_vars = 10294 × 1000
    obs: 'batch', 'study', 'cell_type', 'size_factors'
```

[10]:
```
target_adata
```

[10]:
```
View of AnnData object with n_obs × n_vars = 5387 × 1000
    obs: 'batch', 'study', 'cell_type', 'size_factors'
```

5.10.3 Create TRVAE model and train it on reference dataset

[11]:
```
trvae = sca.models.TRVAE(
    adata=source_adata,
    condition_key=condition_key,
    conditions=source_conditions,
    hidden_layer_sizes=[128, 128],
)
```

```
INITIALIZING NEW NETWORK...
Encoder Architecture:
        Input Layer in, out and cond: 1000 128 3
        Hidden Layer 1 in/out: 128 128
        Mean/Var Layer in/out: 128 10
Decoder Architecture:
        First Layer in, out and cond:  10 128 3
        Hidden Layer 1 in/out: 128 128
        Output Layer in/out:  128 1000
```

[12]:
```
trvae.train(
    n_epochs=trvae_epochs,
    alpha_epoch_anneal=200,
    early_stopping_kwargs=early_stopping_kwargs
)
```

```
Trying to set attribute `.obs` of view, copying.
Trying to set attribute `.obs` of view, copying.
```

```
Valid_data 1029
Condition: 0 Counts in TrainData: 821
Condition: 1 Counts in TrainData: 147
Condition: 2 Counts in TrainData: 61
 |------------| 40.6%  - epoch_loss:    2387 - epoch_unweighted_loss:    2387 - epoch_
↪recon_loss:    2367 - epoch_kl_loss:      18 - epoch_mmd_loss:       2 - val_loss: ␣
↪  1267 - val_unweighted_loss:    1267 - val_recon_loss:    1248 - val_kl_loss:
↪13 - val_mmd_loss:       5
```

(continues on next page)

(continued from previous page)

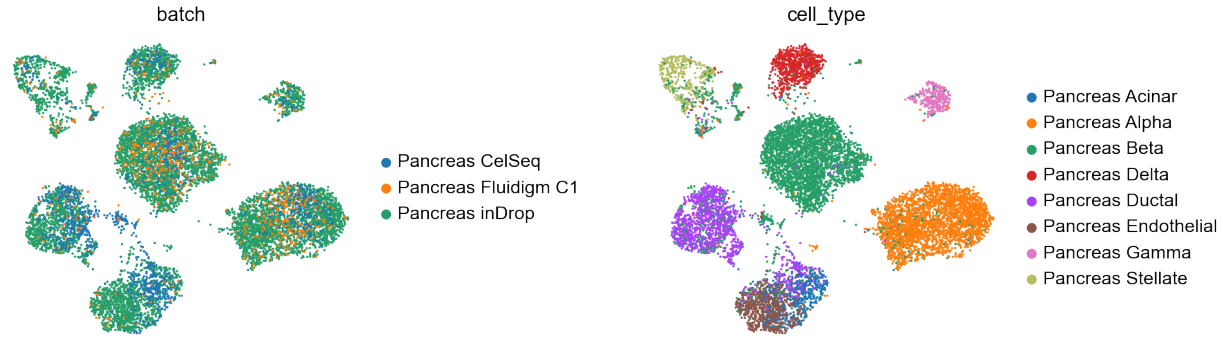
```
ADJUSTED LR
 |-----------| 47.0%  - epoch_loss:    2357 - epoch_unweighted_loss:    2357 - epoch_
↪recon_loss:    2338 - epoch_kl_loss:      17 - epoch_mmd_loss:       2 - val_loss:
↪  1346 - val_unweighted_loss:    1346 - val_recon_loss:    1327 - val_kl_loss:
↪14 - val_mmd_loss:       5
ADJUSTED LR
 |-----------| 48.4%  - epoch_loss:    2371 - epoch_unweighted_loss:    2371 - epoch_
↪recon_loss:    2352 - epoch_kl_loss:      18 - epoch_mmd_loss:       2 - val_loss:
↪  1302 - val_unweighted_loss:    1302 - val_recon_loss:    1284 - val_kl_loss:
↪14 - val_mmd_loss:       5
Stopping early: no improvement of more than 0 nats in 20 epochs
If the early stopping criterion is too strong, please instantiate it with different
↪parameters in the train method.
Saving best state of network...
Best State was in Epoch 220
```

5.10.4 Create anndata file of latent representation and compute UMAP

[13]:
```
adata_latent = sc.AnnData(trvae.get_latent())
adata_latent.obs['cell_type'] = source_adata.obs[cell_type_key].tolist()
adata_latent.obs['batch'] = source_adata.obs[condition_key].tolist()
```

[14]:
```
sc.pp.neighbors(adata_latent, n_neighbors=8)
sc.tl.leiden(adata_latent)
sc.tl.umap(adata_latent)
sc.pl.umap(adata_latent,
           color=['batch', 'cell_type'],
           frameon=False,
           wspace=0.6,
           )
```

```
... storing 'cell_type' as categorical
... storing 'batch' as categorical
```



After pretraining the model can be saved for later use

[15]:
```
ref_path = 'reference_model/'
trvae.save(ref_path, overwrite=True)
```

5.10.5 Perform surgery on reference model and train on query dataset

[16]:
```
new_trvae = trvae.load_query_data(adata=target_adata, reference_model=ref_path)
```

```
INITIALIZING NEW NETWORK...
Encoder Architecture:
        Input Layer in, out and cond: 1000 128 5
        Hidden Layer 1 in/out: 128 128
        Mean/Var Layer in/out: 128 10
Decoder Architecture:
        First Layer in, out and cond:  10 128 5
        Hidden Layer 1 in/out: 128 128
        Output Layer in/out:  128 1000
```

[17]:
```
new_trvae.train(
    n_epochs=surgery_epochs,
    alpha_epoch_anneal=200,
    early_stopping_kwargs=early_stopping_kwargs,
    weight_decay=0
)
```

```
Trying to set attribute `.obs` of view, copying.
Trying to set attribute `.obs` of view, copying.
Valid_data 538
Condition: 0 Counts in TrainData: 0
Condition: 1 Counts in TrainData: 0
 |--------| 64.0%  - epoch_loss:    2664 - epoch_unweighted_loss:    2664 - epoch_
↪recon_loss:    2647 - epoch_kl_loss:      16 - epoch_mmd_loss:       0 - val_loss:
↪  2606 - val_unweighted_loss:    2606 - val_recon_loss:    2589 - val_kl_loss:
↪16 - val_mmd_loss:       1
ADJUSTED LR
 |-------| 68.8%  - epoch_loss:    2576 - epoch_unweighted_loss:    2576 - epoch_
↪recon_loss:    2559 - epoch_kl_loss:      16 - epoch_mmd_loss:       0 - val_loss:
↪  2493 - val_unweighted_loss:    2493 - val_recon_loss:    2477 - val_kl_loss:
↪16 - val_mmd_loss:       1
ADJUSTED LR
 |------| 70.2%  - epoch_loss:    2528 - epoch_unweighted_loss:    2528 - epoch_recon_
↪loss:    2512 - epoch_kl_loss:      16 - epoch_mmd_loss:       0 - val_loss:
↪2495 - val_unweighted_loss:    2495 - val_recon_loss:    2478 - val_kl_loss:
↪16 - val_mmd_loss:       1
Stopping early: no improvement of more than 0 nats in 20 epochs
If the early stopping criterion is too strong, please instantiate it with different
↪parameters in the train method.
Saving best state of network...
Best State was in Epoch 329
```

[18]:
```
adata_latent = sc.AnnData(new_trvae.get_latent())
adata_latent.obs['cell_type'] = target_adata.obs[cell_type_key].tolist()
adata_latent.obs['batch'] = target_adata.obs[condition_key].tolist()
```

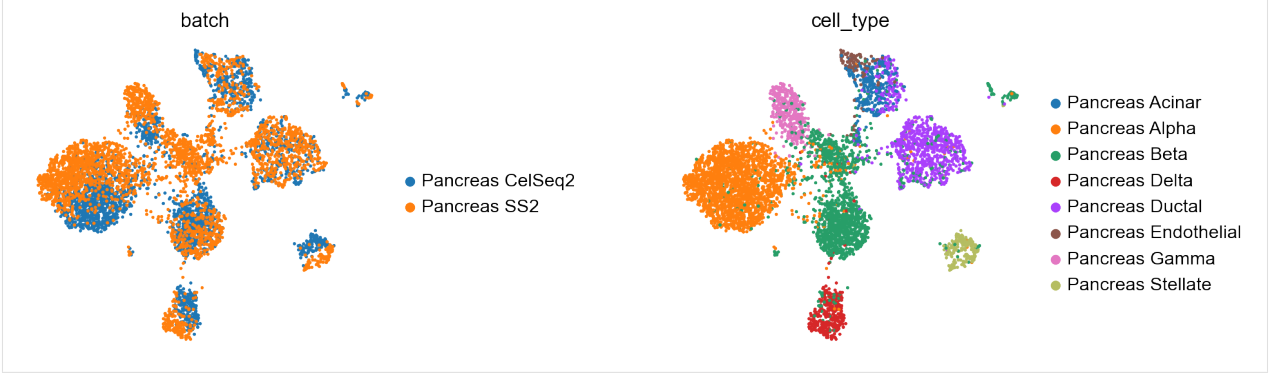
[19]:
```
sc.pp.neighbors(adata_latent, n_neighbors=8)
sc.tl.leiden(adata_latent)
sc.tl.umap(adata_latent)
sc.pl.umap(adata_latent,
           color=['batch', 'cell_type'],
```

(continues on next page)

(continued from previous page)

```
            frameon=False,
            wspace=0.6,
            )
```

```
... storing 'cell_type' as categorical
... storing 'batch' as categorical
```



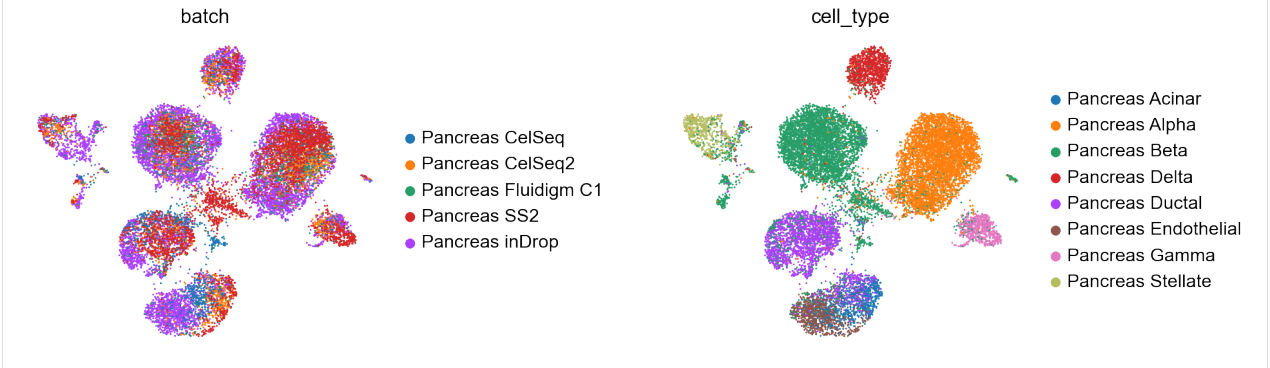
[20]:
```
surg_path = 'surgery_model'
new_trvae.save(surg_path, overwrite=True)
```

5.10.6 Get latent representation of reference + query dataset and compute UMAP

[21]:
```
full_latent = sc.AnnData(new_trvae.get_latent(adata.X, adata.obs[condition_key]))
full_latent.obs['cell_type'] = adata.obs[cell_type_key].tolist()
full_latent.obs['batch'] = adata.obs[condition_key].tolist()
```

[22]:
```
sc.pp.neighbors(full_latent, n_neighbors=8)
sc.tl.leiden(full_latent)
sc.tl.umap(full_latent)
sc.pl.umap(full_latent,
           color=['batch', 'cell_type'],
           frameon=False,
           wspace=0.6,
           )
```

```
... storing 'cell_type' as categorical
... storing 'batch' as categorical
```



PYTHON MODULE INDEX

S

INDEX

R

S

T

U